

# Motor Control Blockset™

Reference



# MATLAB® & SIMULINK®

R2022a



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

## *Motor Control Blockset™ Reference*

© COPYRIGHT 2020–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

## **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## **Revision History**

March 2020	Online only	New for Version 1.0 (Release 2020a)
September 2020	Online only	Revised for Version 1.1 (Release R2020b)
March 2021	Online only	Revised for Version 1.2 (Release R2021a)
September 2021	Online only	Revised for Version 1.3 (Release R2021b)
March 2022	Online only	Revised for Version 1.4 (Release R2022a)

<b>1</b>	<b>Blocks</b>
----------	---------------



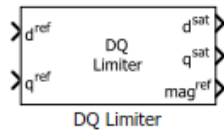
# Blocks

---

## DQ Limiter

Saturate voltages (or current) in the  $dq$  reference frame

**Library:** Motor Control Blockset / Controls / Control Reference



### Description

The DQ Limiter block generates saturated values of the input voltages (or current) in the  $dq$  reference frame, based on the phase voltage (or current) peak limit of the inverter.

The block accepts reference values of  $d$  and  $q$  axis voltages (or current) and outputs the corresponding saturated values. The block also provides the unsaturated peak value of the reference  $dq$  voltages (or current) to enable field weakening control.

### Equations

These equations describe the computation of saturated  $dq$  voltage (or current) values by the block.

$$mag^{ref} = \sqrt{(d^{ref})^2 + (q^{ref})^2}$$

When  $mag^{ref} \leq D - Q$  saturation limit.

- $d^{sat} = d^{ref}$
- $q^{sat} = q^{ref}$

When  $mag^{ref} > D - Q$  saturation limit.

- $d^{sat} = \frac{d^{ref}}{\sqrt{(d^{ref})^2 + (q^{ref})^2}} \times D - Q$  saturation limit
- $q^{sat} = \frac{q^{ref}}{\sqrt{(d^{ref})^2 + (q^{ref})^2}} \times D - Q$  saturation limit

### Ports

#### Input

**$d^{ref}$  — Reference  $d$ -axis voltage (or current)**  
scalar

Reference voltage (or current) value along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

**$q^{ref}$  — Reference  $q$ -axis voltage (or current)**  
scalar

Reference voltage (or current) value along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

### Output

#### **$d^{\text{sat}}$ — Saturated $d$ -axis voltage (or current)**

scalar

Saturated voltage (or current) value along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$q^{\text{sat}}$ — Saturated $q$ -axis voltage (or current)**

scalar

Saturated voltage (or current) value along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$\text{mag}^{\text{ref}}$ — Unsaturated peak value of block inputs**

scalar

Unsaturated peak value of the input voltages (or current).

Data Types: `single` | `double` | `fixed point`

## Parameters

#### **D-Q saturation limit — Phase voltage (or current) peak limit**

1 (default) | scalar

The maximum magnitude of the stator phase voltage (or current) that the inverter can supply to the motor.

*D – Q saturation limit (current)* is usually the rated current of the motor. When you work with the Per-Unit system (PU), you should convert the rated current of the motor to Per-Unit value with respect to the base current.

*D – Q saturation limit (voltage)* is the maximum phase voltage supplied by the inverter. Generally it is  $\frac{V_{\text{dc}}}{\sqrt{3}}$  for Space Vector PWM and  $\frac{V_{\text{dc}}}{2}$  for Sinusoidal PWM, where  $V_{\text{dc}}$  is the DC link voltage of the inverter.

---

**Note** You can enter either per unit or SI unit voltage (or current) value in this parameter. For optimum performance, we recommend that you provide a per unit value.

---

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

Discrete PI Controller with anti-windup and reset | ACIM Feed Forward Control | Inverse Park Transform | MTPA Control Reference | Vector Control Reference

## **Topics**

“Open-Loop and Closed-Loop Control”  
“Field-Oriented Control (FOC)”

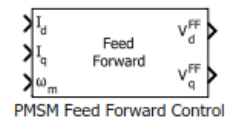
**Introduced in R2020a**



# PMSM Feed Forward Control

Decouple  $d$ -axis and  $q$ -axis current to eliminate disturbance

**Library:** Motor Control Blockset / Controls / Control Reference



## Description

The PMSM Feed Forward Control block decouples  $d$ -axis and  $q$ -axis current controls and generates the corresponding feed-forward voltage gains to enable field-oriented control of Permanent Magnet Synchronous Motor (PMSM).

The block accepts feedback values of  $d$ -axis and  $q$ -axis currents and the mechanical speed of the rotor.

## Equations

If you select Per-Unit (PU) in the **Input units** parameter, the block converts the inputs to SI units before performing any computation. After calculating the output, the block converts the values back to per unit values.

These equations describe the computation of feed-forward gain by the block.

$$V_d^{FF} = -\omega_e \lambda_q = -\omega_e L_q I_q$$

$$V_q^{FF} = \omega_e \lambda_d = \omega_e L_d I_d + \omega_e \lambda_m$$

For detailed set of equations and assumptions that Motor Control Blockset uses for a PMSM, see "Mathematical Model of PMSM" on page 1-127.

where:

- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $L_d$  and  $L_q$  are the  $d$ -axis and  $q$ -axis stator winding inductances (Henry).
- $I_d$  and  $I_q$  are the  $d$ -axis and  $q$ -axis currents (Amperes).
- $\lambda_d$  and  $\lambda_q$  are the magnetic fluxes along the  $d$ - and  $q$ -axes (Weber).
- $\lambda_m$  is the permanent magnet flux linkage (Weber).

## Ports

### Input

#### $I_d$ — D-axis current

scalar

Current along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

 **$I_q$  — Q-axis current**

scalar

Current along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

 **$\omega_m$  — Mechanical speed of rotor**

scalar

Mechanical speed of the rotor.

Data Types: `single` | `double` | `fixed point`

**Output** **$V_d^{FF}$  — D-axis feed-forward voltage gain**

scalar

Feed-forward voltage gain along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

 **$V_q^{FF}$  — Q-axis feed-forward voltage gain**

scalar

Feed-forward voltage gain along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

**Parameters****Number of pole pairs — Number of pole pairs available in motor**

4 (default) | scalar

Number of pole pairs available in the motor.

**Stator d-axis inductance (H) — D-axis stator winding inductance**

$0.2e-3$  (default) | scalar

Stator winding inductance (in Henry) along the direct-axis of the rotating  $dq$  reference frame.

**Stator q-axis inductance (H) — Q-axis stator winding inductance**

$0.2e-3$  (default) | scalar

Stator winding inductance (in Henry) along the quadrature-axis of the rotating  $dq$  reference frame.

**Permanent magnet flux linkage (Wb) — PM flux linkage**

$6.4e-3$  (default) | scalar

Peak permanent magnet flux linkage (in Weber).

**Output Saturation (V) — Saturation limit for output values**

$24/\sqrt{3}$  (default) | scalar

Saturation limit (in Volts) for the output voltages  $V_d^{FF}$  and  $V_q^{FF}$ .

**Input units – Unit of input values**

Per-Unit (PU) (default) | SI Units

Unit of the input values.

**Base Voltage (V) – Nominal voltage limit**

24/sqrt(3) (default) | scalar

Base voltage (in Volts) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Base Current (A) – Nominal current limit**

19.3 (default) | scalar

Base current (in Amperes) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Base Speed (rpm) – Nominal speed limit**

4107 (default) | scalar

Base speed (in rpm) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

ACIM Feed Forward Control | Park Transform | Speed Measurement | Discrete PI Controller with anti-windup and reset | DQ Limiter

**Topics**

“Open-Loop and Closed-Loop Control”

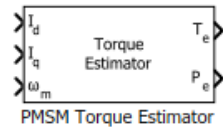
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

# PMSM Torque Estimator

Estimate electromechanical torque and power

**Library:** Motor Control Blockset / Controls / Control Reference



## Description

The PMSM Torque Estimator block generates electromechanical torque and power estimates to enable field-oriented control of Permanent Magnet Synchronous Motor (PMSM). The block outputs the mathematically computed electromechanical torque for constant motor parameters. To measure an accurate torque value, we recommend that you use a physical sensor.

The block accepts feedback values of  $d$ - and  $q$ -axis current and mechanical speed as inputs.

## Equations

If you select Per-Unit (PU) in the **Input units** parameter, the block converts the inputs to SI units before performing any computation. After calculating the output, the block converts the output back to per unit values.

These equations describe the computation of electromechanical torque and power estimates by the block.

$$T_e = \frac{3}{2}p\{\lambda_m I_q + (L_d - L_q)I_d I_q\}$$

$$P_e = T_e \cdot \omega_m$$

For detailed set of equations and assumptions that Motor Control Blockset uses for a PMSM, see “Mathematical Model of PMSM” on page 1-127.

where:

- $L_d$  and  $L_q$  are the  $d$ -axis and  $q$ -axis stator winding inductances (Henry).
- $I_d$  and  $I_q$  are the  $d$ -axis and  $q$ -axis current (Amperes).
- $\lambda_m$  is the permanent magnet flux linkage (Weber).
- $p$  is the number of pole pairs available in the motor.
- $\omega_m$  is the mechanical speed of the rotor (Radians/ sec).

## Ports

### Input

**$I_d$  — D-axis current**

scalar

Current along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

### **$I_q$ — Q-axis current**

scalar

Current along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

### **$\omega_m$ — Mechanical speed of rotor**

scalar

Mechanical speed of the rotor.

Data Types: single | double | fixed point

## **Output**

### **$T_e$ — Electromechanical torque**

scalar

Electromechanical torque of the rotor.

Data Types: single | double | fixed point

### **$P_e$ — Electromechanical power**

scalar

Electromechanical power of the rotor.

Data Types: single | double | fixed point

## **Parameters**

### **Number of pole pairs — Number of pole pairs available in motor**

4 (default) | scalar

Number of pole pairs available in the motor.

### **Stator d-axis inductance (H) — D-axis stator winding inductance**

0.2e-3 (default) | scalar

Stator winding inductance (henry) along the direct-axis of the rotating  $dq$  reference frame.

### **Stator q-axis inductance (H) — Q-axis stator winding inductance**

0.2e-3 (default) | scalar

Stator winding inductance (henry) along the quadrature-axis of the rotating  $dq$  reference frame.

### **Permanent magnet flux linkage (Wb) — Permanent magnet flux linkage**

6.4e-3 (default) | scalar

Peak permanent magnet flux linkage (weber).

### **Input units — Unit of input values**

Per-Unit (PU) (default) | SI Units

Unit of the input values.

**Base Voltage (V) — Nominal voltage limit**

24/sqrt(3) (default) | scalar

Base voltage (in Volts) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Base Current (A) — Nominal current limit**

19.3 (default) | scalar

Base current (in Amperes) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Base Speed (rpm) — Nominal speed limit**

4107 (default) | scalar

Base speed (in rpm) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Base torque (Nm) — Nominal torque limit**

0.74112 (default) | scalar

Base torque (in Nm) for per-unit system. See “Per-Unit System” page for more details.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Dependencies**

To display this parameter, set **Input units** to Per-Unit (PU).

**Base power (W) — Nominal power limit**

401.143 (default) | scalar

Base power (in W) for per-unit system. See “Per-Unit System” page for more details.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Dependencies**

To display this parameter, set **Input units** to Per-Unit (PU).

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

ACIM Torque Estimator | Park Transform | Speed Measurement | MTPA Control Reference | Vector Control Reference

**Topics**

“Open-Loop and Closed-Loop Control”

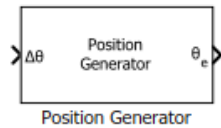
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

# Position Generator

Generate position ramp of fixed frequency

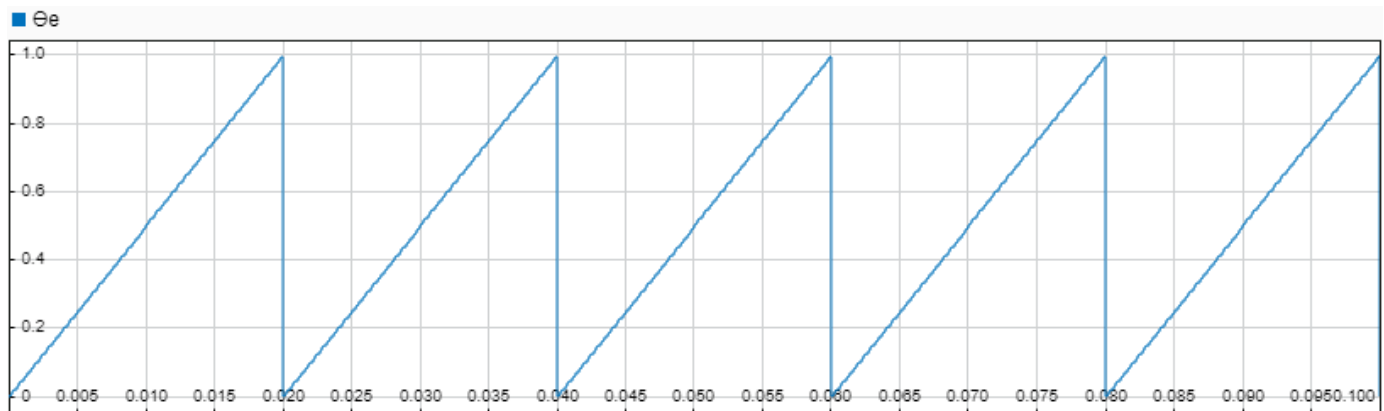
**Library:** Motor Control Blockset / Controls / Control Reference



## Description

The Position Generator block generates a position ramp signal (with a frequency that is identical to that of the reference voltage signal) using the position increment value of the reference signal.

We recommend that you use fixed-step discrete solver for this block to enable code generation and ensure accurate simulation.



## Ports

### Input

#### $\Delta\theta$ — Position increment value

scalar

Position increment value of a fixed frequency reference voltage signal (in either per unit, radians, or degrees). These equations describe how the block computes the position increment:

- $\Delta\theta$  (per unit) = Frequency  $\times$  Sample Time
- $\Delta\theta$  (radians) =  $2\pi \times$  Frequency  $\times$  Sample Time
- $\Delta\theta$  (degrees) =  $360 \times$  Frequency  $\times$  Sample Time

Data Types: single | double | fixed point

#### Reset — External reset signal

scalar



External pulse that resets the position ramp output based on the value of the **External reset** parameter.

### Dependencies

To enable this port, set **External reset** to either active high resets to zero or active high resets to initial condition.

Data Types: `single` | `double` | `fixed point`

### Output

#### $\theta_e$ — Reference voltage position

scalar

Position or phase value of the reference voltage signal (in either per unit, radians, or degrees).

Data Types: `single` | `double` | `fixed point`

## Parameters

### Theta Units — Unit of $\theta$

Per-unit (default) | Radians | Degrees

Unit of the input position increment value and the output reference voltage position.

### Initial theta output — Initial value of $\theta_e$

$\theta$  (default) | scalar

Output position ramp value (in either per unit, radians, or degrees) at initial time (0 seconds).

### External reset — Output value on reset

`none` (default) | `active high resets to zero` | `active high resets to initial condition`

Output position ramp value (in either per unit, radians, or degrees) at the time when the block receives an active high external reset pulse. You can reset the output to either zero or to equal the value of the **Initial theta output** parameter.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

ACIM Slip Speed Estimator | Sine-Cosine Lookup | 3-Phase Sine Voltage Generator | Vector Control Reference

### Topics

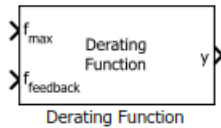
“Open-Loop and Closed-Loop Control”  
 “Field-Oriented Control (FOC)”

**Introduced in R2020a**

## Derating Function

Compute derating factor

**Library:** Motor Control Blockset / Controls / Controllers



### Description

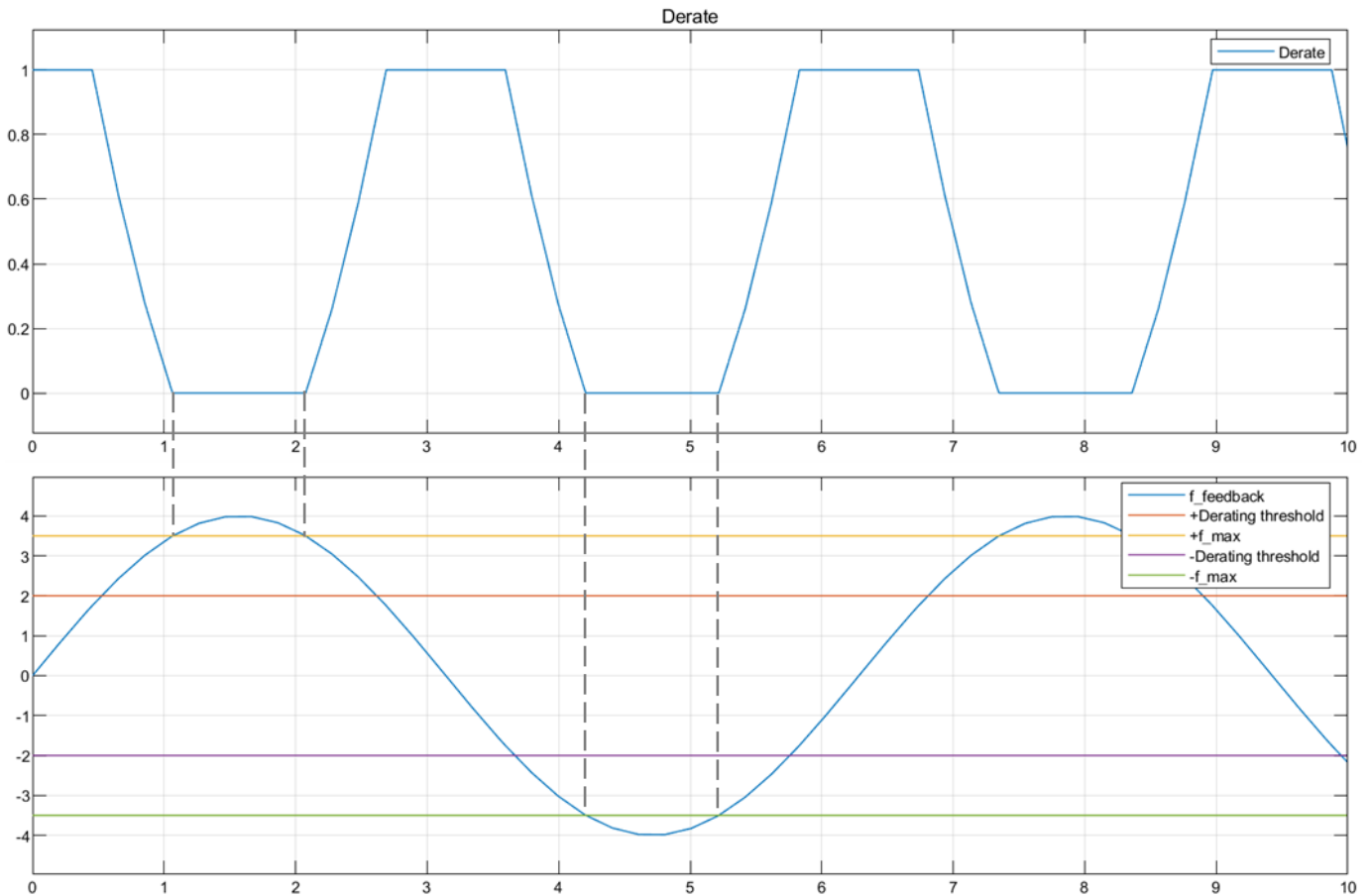
The Derating Function block generates the derating factor ( $y$ ) according to the feedback ( $f_{\text{feedback}}$ ) and maximum limit ( $f_{\text{max}}$ ) values of the input reference signal.

The derating factor:

- Remains equal to one when  $f_{\text{feedback}}$  lies between positive and negative values of the **Derating threshold**. The derating factor varies linearly outside this range according to  $f_{\text{feedback}}$ .
- Remains equal to zero when the reference signal lies beyond (positive or negative)  $f_{\text{max}}$ .

Therefore, you can use the generated derating factor to derate a control signal after the reference signal crosses the specified **Derating threshold**.

This figure shows the block output when you use a sinusoidal wave as  $f_{\text{feedback}}$ .



## Equations

The **Derating threshold** parameter,  $x$  indicates the percentage of peak amplitude for the reference signal. The **Derating threshold** is 0.5 in the block output shown, which results in a threshold value of 2 (for the peak amplitude value of 4 for the sinusoidal reference signal).

$$x = [0, 1)$$

This equation describes how the block computes the derating factor ( $y$ ).

$$\text{Derating factor } (y) = 1 - \frac{f_{\text{feedback}} - x f_{\text{max}}}{(1 - x) f_{\text{max}}}$$

## Ports

### Input

**$f_{\text{max}}$  — Maximum reference signal limit**

scalar

Maximum limit of the reference signal value beyond which the derating factor becomes zero.

Data Types: single | double | fixed point

**$f_{\text{feedback}}$  — Reference feedback signal**  
scalar

Reference signal that the block uses to generate the derating factor, which you can then use to derate a control signal.

Data Types: `single` | `double` | `fixed point`

## Output

**$y$  — Derating factor**  
scalar

Derating factor that the block generates based on the feedback and maximum limit values of the reference signal when the signal exceeds the value of the **Derating threshold** parameter.

Data Types: `single` | `double` | `fixed point`

## Parameters

**Derating threshold — Threshold beyond which derating must occur**  
0.9 (default) | scalar in the range [0, 1)

The reference signal value beyond which the block generates the derating factor.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

[Speed Measurement](#) | [MTPA Control Reference](#) | [Vector Control Reference](#) | [PMSM Torque Estimator](#)

## Topics

“Open-Loop and Closed-Loop Control”

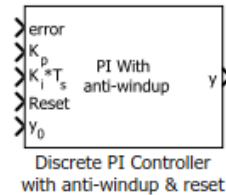
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

## Discrete PI Controller with anti-windup and reset

Implement discrete PI controller with anti-windup and reset functionality

**Library:** Motor Control Blockset / Controls / Controllers



### Description

The Discrete PI Controller with anti-windup and reset block performs discrete-time PI controller computation using the error signal and proportional and integral gain inputs. The error signal is the difference between the reference signal and the measured feedback. The block outputs a weighted sum of the input error signal and the integral of the input error signal.

You can tune the Discrete PI Controller coefficients ( $K_p$  and  $K_i$ ) either manually or automatically. Automatic tuning requires Simulink® Control Design™ software.

The block also supports anti-windup functionality, which makes the block output to comply with the register size of the processor. You can reset the integrator to the initial condition ( $y_0$ ).

We recommend that you use fixed-step discrete solver for this block to enable code generation and ensure accurate simulation.

### Ports

#### Input

##### **error** — Variation of system output from expected value

scalar

Difference between a reference signal and the system output.

Data Types: single | double | fixed point

##### **$K_p$** — Proportional gain

scalar

Proportional gain value that you can compute either manually or automatically.

Data Types: single | double | fixed point

##### **$K_i * T_s$** — Integral gain pre-multiplied by integrator sample time

scalar

Integral gain input that you can compute either manually or automatically. You must premultiply the integral gain value by the integrator sample time ( $T_s$ ) for the block to execute within asynchronous interrupts.

Data Types: `single` | `double` | `fixed point`

### **Reset — External reset signal**

scalar

External pulse that resets the block output to the value of the initial output from the integrator ( $y_0$ ).

Data Types: `single` | `double` | `fixed point`

### **$y_0$ — Value of initial output from integrator**

scalar

Initial value of the integrator or block output after receiving a reset pulse.

Data Types: `single` | `double` | `fixed point`

### **Output**

#### **$y$ — PI controller output**

scalar

Control signal that is identical to the reference signal.

Data Types: `single` | `double` | `fixed point`

## **Parameters**

### **Upper saturation limit — Upper limit for block output**

1 (default) | scalar

The block holds the output at the **Upper saturation limit** whenever the weighted sum of the proportional and integral actions exceeds this value.

### **Lower saturation limit — Lower limit for block output**

-1 (default) | scalar

The block holds the output at the **Lower saturation limit** whenever the weighted sum of the proportional and integral actions goes below this value.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

Discrete PI Controller | Park Transform | Speed Measurement | DQ Limiter | ACIM Control Reference | MTPA Control Reference

### **Topics**

“Open-Loop and Closed-Loop Control”

“Field-Oriented Control (FOC)”

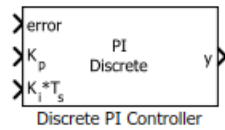
**Introduced in R2020a**



# Discrete PI Controller

Implement discrete PI controller

**Library:** Motor Control Blockset / Controls / Controllers



## Description

The Discrete PI Controller block performs discrete-time PI controller computation using the error signal and proportional and integral gain inputs. The error signal is the difference between the reference signal and the measured feedback. The block outputs a weighted sum of the input error signal and the integral of the input error signal.

You can tune the Discrete PI Controller coefficients ( $K_p$  and  $K_i$ ) either manually or automatically. Automatic tuning requires Simulink Control Design™ software.

We recommend that you use fixed-step discrete solver for this block to enable code generation and ensure accurate simulation.

## Ports

### Input

**error — Variation of system output from expected value**

scalar

Difference between a reference signal and the system output.

Data Types: `single` | `double` | `fixed point`

**$K_p$  — Proportional gain**

scalar

Proportional gain value that you can compute either manually or automatically.

Data Types: `single` | `double` | `fixed point`

**$K_i * T_s$  — Integral gain pre-multiplied by integrator sample time**

scalar

Integral gain input that you can compute either manually or automatically. You must premultiply the integral gain value by the integrator sample time ( $T_s$ ) for the block to execute within asynchronous interrupts.

Data Types: `single` | `double` | `fixed point`

### Output

**y — PI controller output**

scalar

Control signal that is identical to the reference signal.

Data Types: `single` | `double` | `fixed point`

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

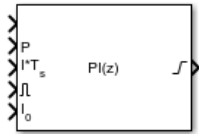
Discrete PI Controller with anti-windup and reset

## **Introduced in R2020a**

# PI Controller

Discrete-time PID Controller

**Library:** Motor Control Blockset / Controls / Controllers



## Description

The PI Controller block implements a discrete-time PID controller (PID, PI, PD, P only, or I only). The block is identical to the Discrete PID Controller Simulink block.

The block output is a weighted sum of the input signal, the integral of the input signal, and the derivative of the input signal. The weights are the proportional, integral, and derivative gain parameters. A first-order pole filters the derivative action.

- Controller type (PID, PI, PD, P only, or I only) — See the **Controller** parameter.
- Controller form (Parallel or Ideal) — See the **Form** parameter.
- Time domain (continuous or discrete) — See the **Time domain** parameter.
- Initial conditions and reset trigger — See the **Source** and **External reset** parameters.
- Output saturation limits and built-in anti-windup mechanism — See the **Limit output** parameter.
- Signal tracking for bumpless control transfer and multiloop control — See the **Enable tracking mode** parameter.

As you change these options, the internal structure of the block changes by activating different variant subsystems. (For more information, see “Variant Subsystems”). To examine the internal structure of the block and its variant subsystems, right-click the block and select **Mask > Look Under Mask**.

## PID Gain Tuning

The PID controller gains are tunable either manually or automatically. Automatic tuning requires Simulink Control Design™ software. For more information about automatic tuning, see the **Select tuning method** parameter.

## Ports

### Input

**Port\_1( u ) — Error signal input**

scalar | vector

Difference between a reference signal and the output of the system under control.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

## P — Proportional gain

scalar | vector

Proportional gain, provided from a source external to the block. External gain input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use external gain input to implement gain-scheduled PID control. In gain-scheduled control, you determine the PID coefficients by logic or other calculations in your model and feed them to the block.

### Dependencies

To enable this port, set **Controller parameters Source** to `external`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

## I\*T<sub>s</sub> — Integral gain multiplied by sample time

scalar | vector

Integral gain multiplied by the controller sample time, provided from a source external to the block. External gain input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use external gain input to implement gain-scheduled PID control. In gain-scheduled control, you determine the PID coefficients by logic or other calculations in your model and feed them to the block.

---

**Note** PID tuning tools, such as the **PID Tuner** app and Closed-Loop PID Autotuner block, tune the gain **I** but not **I\*T<sub>s</sub>**. Therefore, multiply the integral gain value you obtain from a tuning tool by the sample time before you supply it to this port.

---

When you use **I\*T<sub>s</sub>** instead of **I**, the block requires fewer calculations to perform integration. This improves the execution time of the generated code.

### Dependencies

To enable this port, set **Controller parameters Source** to `external`, set **Controller** to a controller type that has integral action, and enable the **Use I\*T<sub>s</sub>** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

## I — Integral gain

scalar | vector

Integral gain, provided from a source external to the block. External gain input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use external gain input to implement gain-scheduled PID control. In gain-scheduled control, you determine the PID coefficients by logic or other calculations in your model and feed them to the block.

When you supply gains externally, time variations in the integral gain are also integrated. This result occurs because of the way the PID gains are implemented within the block. For details, see the **Controller parameters Source** parameter.

**Dependencies**

To enable this port, set **Controller parameters Source** to `external`, and set **Controller** to a controller type that has integral action.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

**D – Derivative gain**

`scalar` | `vector`

Derivative gain, provided from a source external to the block. External gain input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use external gain input to implement gain-scheduled PID control. In gain-scheduled control, you determine the PID coefficients by logic or other calculations in your model and feed them to the block.

When you supply gains externally, time variations in the derivative gain are also differentiated. This result occurs because of the way the PID gains are implemented within the block. For details, see the **Controller parameters Source** parameter.

**Dependencies**

To enable this port, set **Controller parameters Source** to `external`, and set **Controller** to a controller type that has derivative action.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

**N – Filter coefficient**

`scalar` | `vector`

Derivative filter coefficient, provided from a source external to the block. External coefficient input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use the external input to implement gain-scheduled PID control. In gain-scheduled control, you determine the PID coefficients by logic or other calculations in your model and feed them to the block.

**Dependencies**

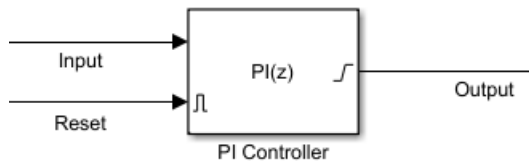
To enable this port, set **Controller parameters Source** to `external`, and set **Controller** to a controller type that has a filtered derivative.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

**Reset – External reset trigger**

`scalar`

Trigger to reset the integrator and filter to their initial conditions. The value of the **External reset** parameter determines whether reset occurs on a rising signal, a falling signal, or a level signal. The port icon indicates the selected trigger type. For example, the following illustration shows a PID block with **External reset** set to `level`.



When the trigger occurs, the block resets the integrator and filter to the initial conditions specified by the **Integrator Initial condition** and **Filter Initial condition** parameters or the **I<sub>0</sub>** and **D<sub>0</sub>** ports.

---

**Note** To be compliant with the Motor Industry Software Reliability Association (MISRA®) software standard, your model must use Boolean signals to drive the external reset ports of the PID controller block.

---

### Dependencies

To enable this port, set **External reset** to any value other than none.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point | Boolean

### I<sub>0</sub> — Integrator initial condition

scalar | vector

Integrator initial condition, provided from a source external to the block.

### Dependencies

To enable this port, set **Initial conditions Source** to external, and set **Controller** to a controller type that has integral action.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

### D<sub>0</sub> — Filter initial condition

scalar | vector

Initial condition of the derivative filter, provided from a source external to the block.

### Dependencies

To enable this port, set **Initial conditions Source** to external, and set **Controller** to a controller type that has derivative action.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

### up — Output saturation upper limit

scalar | vector

Upper limit of the block output, provided from a source external to the block. If the weighted sum of the proportional, integral, and derivative actions exceeds the value provided at this port, the block output is held at that value.

### Dependencies

To enable this port, select **Limit output** and set the output saturation **Source** to external.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

### Lo — Output saturation lower limit

scalar | vector

Lower limit of the block output, provided from a source external to the block. If the weighted sum of the proportional, integral, and derivative actions goes below the value provided at this port, the block output is held at that value.

#### Dependencies

To enable this port, select **Limit output** and set the output saturation **Source** to external.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

### TR — Tracking signal

scalar | vector

Signal for controller output to track. When signal tracking is active, the difference between the tracking signal and the block output is fed back to the integrator input. Signal tracking is useful for implementing bumpless control transfer in systems that switch between two controllers. It can also be useful to prevent block windup in multiloop control systems. For more information, see the **Enable tracking mode** parameter.

#### Dependencies

To enable this port, select the **Enable tracking mode** parameter.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

### T<sub>DTI</sub> — Discrete-integrator time

scalar

Discrete-integrator time, provided as a scalar to the block. You can use your own value of discrete-time integrator sample time that defines the rate at which the block is going to be run either in Simulink or on external hardware. The value of the discrete-time integrator time should match the average sampling rate of the external interrupts, when the block is used inside a conditionally-executed subsystem.

In other words, you can specify  $T_s$  for any of the integrator methods below such that the value matches the average sampling rate of the external interrupts. In discrete time, the derivative term of the controller transfer function is:

$$D\left[\frac{N}{1 + N\alpha(z)}\right],$$

where  $\alpha(z)$  depends on the integrator method you specify with this parameter.

Forward Euler

$$\alpha(z) = \frac{T_s}{z - 1} \cdot$$

Backward Euler

$$\alpha(z) = \frac{T_s z}{z - 1} \cdot$$

Trapezoidal

$$\alpha(z) = \frac{T_s}{2} \frac{z + 1}{z - 1} \cdot$$

For more information about discrete-time integration, see the Discrete-Time Integrator block reference page. For more information on conditionally executed subsystems, see “Conditionally Executed Subsystems Overview”.



## Dependencies

To enable this port, set **Time Domain** to Discrete-time and select the **PID Controller is inside a conditionally executed subsystem** option.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

## Output

### Port\_1( y ) – Controller output

scalar | vector

Controller output, generally based on a sum of the input signal, the integral of the input signal, and the derivative of the input signal, weighted by the proportional, integral, and derivative gain parameters. A first-order pole filters the derivative action. Which terms are present in the controller signal depends on what you select for the **Controller** parameter. The base controller transfer function for the current settings is displayed in the **Compensator formula** section of the block parameters and under the mask. Other parameters modify the block output, such as saturation limits specified by the **Upper Limit** and **Lower Limit** saturation parameters.

The controller output is a vector signal when any of the inputs is a vector signal. In that case, the block acts as  $N$  independent PID controllers, where  $N$  is the number of signals in the input vector.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | fixed point

## Parameters

### Controller – Controller type

PI (default) | PID | PD | P | I

Specify which of the proportional, integral, and derivative terms are in the controller.

#### PID

Proportional, integral, and derivative action.

#### PI

Proportional and integral action only.

#### PD

Proportional and derivative action only.

#### P

Proportional action only.

#### I

Integral action only.

---

**Tip** The controller transfer function for the current setting is displayed in the **Compensator formula** section of the block parameters and under the mask.

---

## Programmatic Use

**Block Parameter:** Controller

**Type:** string, character vector  
**Values:** "PID", "PI", "PD", "P", "I"  
**Default:** "PI"

### Form — Controller structure

Parallel (default) | Ideal

Specify whether the controller structure is parallel or ideal.

#### Parallel

The controller output is the sum of the proportional, integral, and derivative actions, weighted independently by **P**, **I**, and **D**, respectively. For example, for a continuous-time parallel-form PID controller, the transfer function is:

$$C_{par}(s) = P + I\left(\frac{1}{s}\right) + D\left(\frac{Ns}{s+N}\right).$$

For a discrete-time parallel-form controller, the transfer function is:

$$C_{par}(z) = P + I\alpha(z) + D\left[\frac{N}{1+N\beta(z)}\right],$$

where the **Integrator method** and **Filter method** parameters determine  $\alpha(z)$  and  $\beta(z)$ , respectively.

#### Ideal

The proportional gain **P** acts on the sum of all actions. For example, for a continuous-time ideal-form PID controller, the transfer function is:

$$C_{id}(s) = P\left[1 + I\left(\frac{1}{s}\right) + D\left(\frac{Ns}{s+N}\right)\right].$$

For a discrete-time ideal-form controller, the transfer function is:

$$C_{id}(z) = P\left[1 + I\alpha(z) + D\frac{N}{1+N\beta(z)}\right],$$

where the **Integrator method** and **Filter method** parameters determine  $a(z)$  and  $b(z)$ , respectively.

---

**Tip** The controller transfer function for the current settings is displayed in the **Compensator formula** section of the block parameters and under the mask.

---

### Programmatic Use

**Block Parameter:** Controller

**Type:** string, character vector

**Values:** "Parallel", "Ideal"

**Default:** "Parallel"

### Time domain — Specify discrete-time or continuous-time controller

Discrete-time (default) | Continuous-time

When you select **Discrete-time**, it is recommended that you specify an explicit sample time for the block. See the **Sample time (-1 for inherited)** parameter. Selecting **Discrete-time** also enables the **Integrator method**, and **Filter method** parameters.

When the PID Controller block is in a model with synchronous state control (see the State Control block), you cannot select **Continuous-time**.

#### Programmatic Use

**Block Parameter:** TimeDomain

**Type:** string, character vector

**Values:** "Continuous-time", "Discrete-time"

**Default:** "Discrete-time"

#### PID Controller is inside a conditionally executed subsystem — Enable the discrete-integrator time port

off (default) | on

For discrete-time PID controllers, enable the discrete-time integrator port to use your own value of discrete-time integrator sample time. To ensure proper integration, use the  $T_{DTI}$  port to provide a scalar value of  $\Delta t$  for accurate discrete-time integration.

#### Dependencies

To enable this parameter, set **Time Domain** to **Discrete-time**.

#### Programmatic Use

**Block Parameter:** UseExternalTs

**Type:** string, character vector

**Values:** "on", "off"

**Default:** "off"

#### Sample time (-1 for inherited) — Discrete interval between samples

-1 (default) | positive scalar

Specify a sample time by entering a positive scalar value, such as 0.1. The default discrete sample time of -1 means that the block inherits its sample time from upstream blocks. However, it is recommended that you set the controller sample time explicitly, especially if you expect the sample time of upstream blocks to change. The effect of the controller coefficients P, I, D, and N depend on the sample time. Thus, for a given set of coefficient values, changing the sample time changes the performance of the controller.

See “Specify Sample Time” for more information.

To implement a continuous-time controller, set **Time domain** to **Continuous-time**.

---

**Tip** If you want to run the block with an externally specified or variable sample time, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the desired sample time.

---

#### Dependencies

To enable this parameter, set **Time domain** to **Discrete-time**.

**Programmatic Use****Block Parameter:** SampleTime**Type:** scalar**Values:** -1, positive scalar**Default:** -1**Integrator method — Method for computing integral in discrete-time controller**

Forward Euler (default) | Backward Euler | Trapezoidal

In discrete time, the integral term of the controller transfer function is  $I\alpha(z)$ , where  $\alpha(z)$  depends on the integrator method you specify with this parameter.

**Forward Euler**

Forward rectangular (left-hand) approximation,

$$\alpha(z) = \frac{T_s}{z-1}.$$

This method is best for small sampling times, where the Nyquist limit is large compared to the bandwidth of the controller. For larger sampling times, the Forward Euler method can result in instability, even when discretizing a system that is stable in continuous time.

**Backward Euler**

Backward rectangular (right-hand) approximation,

$$\alpha(z) = \frac{T_s z}{z-1}.$$

An advantage of the Backward Euler method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result.

**Trapezoidal**

Bilinear approximation,

$$\alpha(z) = \frac{T_s z + 1}{2 z - 1}.$$

An advantage of the Trapezoidal method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Of all available integration methods, the Trapezoidal method yields the closest match between frequency-domain properties of the discretized system and the corresponding continuous-time system.

---

**Tip** The controller formula for the current setting is displayed in the **Compensator formula** section of the block parameters and under the mask.

---

**Note** For the BackwardEuler or Trapezoidal methods, you cannot generate HDL code for the block if either:

- **Limit output** is selected and **Anti-Windup Method** is anything other than none.
  - **Enable tracking mode** is selected.
-

For more information about discrete-time integration, see the Discrete-Time Integrator block reference page.

### Dependencies

To enable this parameter, set **Time Domain** to Discrete-time and set **Controller** to a controller type with integral action.

### Programmatic Use

**Block Parameter:** IntegratorMethod

**Type:** string, character vector

**Values:** "Forward Euler", "Backward Euler", "Trapezoidal"

**Default:** "Forward Euler"

### Filter method — Method for computing derivative in discrete-time controller

Forward Euler (default) | Backward Euler | Trapezoidal

In discrete time, the derivative term of the controller transfer function is:

$$D\left[\frac{N}{1 + N\alpha(z)}\right],$$

where  $\alpha(z)$  depends on the filter method you specify with this parameter.

#### Forward Euler

Forward rectangular (left-hand) approximation,

$$\alpha(z) = \frac{T_s}{z - 1}.$$

This method is best for small sampling times, where the Nyquist limit is large compared to the bandwidth of the controller. For larger sampling times, the Forward Euler method can result in instability, even when discretizing a system that is stable in continuous time.

#### Backward Euler

Backward rectangular (right-hand) approximation,

$$\alpha(z) = \frac{T_s z}{z - 1} \cdot$$

An advantage of the Backward Euler method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result.

#### Trapezoidal

Bilinear approximation,

$$\alpha(z) = \frac{T_s z + 1}{2 z - 1} \cdot$$

An advantage of the Trapezoidal method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Of all available integration methods, the Trapezoidal method yields the closest match between frequency-domain properties of the discretized system and the corresponding continuous-time system.

---

**Tip** The controller formula for the current setting is displayed in the **Compensator formula** section of the block parameters and under the mask.

---

For more information about discrete-time integration, see the Discrete-Time Integrator block reference page.

#### Dependencies

To enable this parameter, set **Time Domain** to Discrete-time and enable **Use filtered derivative**.

#### Programmatic Use

**Block Parameter:** FilterMethod

**Type:** string, character vector

**Values:** "Forward Euler", "Backward Euler", "Trapezoidal"

**Default:** "Forward Euler"

## Main

### Source — Source for controller gains and filter coefficient

external (default) | internal

Enabling external inputs for the parameters allows you to compute PID gains and filter coefficients externally to the block and provide them to the block as signal inputs.

#### internal

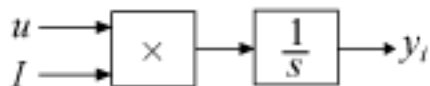
Specify the controller gains and filter coefficient using the block parameters **P**, **I** (or **I\*Ts**), **D**, and **N**.

#### external

Specify the PID gains and filter coefficient externally using block inputs. An additional input port appears on the block for each parameter that is required for the current controller type.

External gain input is useful, for example, when you want to map a different PID parameterization to the PID gains of the block. You can also use external gain input to implement gain-scheduled PID control. In gain-scheduled control, you determine the PID gains by logic or other calculations in your model and feed them to the block.

When you supply gains externally, time variations in the integral and derivative gain values are integrated and differentiated, respectively. This result occurs because in both continuous time and discrete time, the gains are applied to the signal before integration or differentiation. For example, for a continuous-time PID controller with external inputs, the integrator term is implemented as shown in the following illustration.



Within the block, the input signal  $u$  is multiplied by the externally supplied integrator gain,  $I$ , before integration. This implementation yields:

$$y_i = \int uI dt.$$

Thus, the integrator gain is included in the integral. Similarly, in the derivative term of the block, multiplication by the derivative gain precedes the differentiation, which causes the derivative gain  $D$  to be differentiated.

### Programmatic Use

**Block Parameter:** ControllerParametersSource

**Type:** string, character vector

**Values:** "internal", "external"

**Default:** "external"

### Proportional (P) — Proportional gain

1 (default) | scalar | vector

Specify a finite, real gain value for the proportional gain. When **Controller form** is:

- **Parallel** — Proportional action is independent of the integral and derivative actions. For instance, for a continuous-time parallel PID controller, the transfer function is:

$$C_{par}(s) = P + I\left(\frac{1}{s}\right) + D\left(\frac{Ns}{s+N}\right).$$

For a discrete-time parallel-form controller, the transfer function is:

$$C_{par}(z) = P + I\alpha(z) + D\left[\frac{N}{1+N\beta(z)}\right],$$

where the **Integrator method** and **Filter method** parameters determine  $\alpha(z)$  and  $\beta(z)$ , respectively.

- **Ideal** — The proportional gain multiplies the integral and derivative terms. For instance, for a continuous-time ideal PID controller, the transfer function is:

$$C_{id}(s) = P\left[1 + I\left(\frac{1}{s}\right) + D\left(\frac{Ns}{s+N}\right)\right].$$

For a discrete-time ideal-form controller, the transfer function is:

$$C_{id}(z) = P\left[1 + I\alpha(z) + D\frac{N}{1+N\beta(z)}\right],$$

where the **Integrator method** and **Filter method** parameters determine  $\alpha(z)$  and  $\beta(z)$ , respectively.

**Tunable:** Yes

**Dependencies**

To enable this parameter, in the **Main** tab, set the controller-parameters **Source** to `internal` and set **Controller** to `PID`, `PD`, `PI`, or `P`.

**Programmatic Use**

**Block Parameter:** `P`

**Type:** scalar, vector

**Default:** 1

**Integral (I) — Integral gain**

1 (default) | scalar | vector

Specify a finite, real gain value for the integral gain.

**Tunable:** Yes

**Dependencies**

To enable this parameter, in the **Main** tab, set the controller-parameters **Source** to `internal`, and set **Controller** to a type that has integral action.

**Programmatic Use**

**Block Parameter:** `I`

**Type:** scalar, vector



**Default:** 1

### **Integral (I\*Ts) — Integral gain multiplied by sample time**

1 (default) | scalar | vector

Specify a finite, real gain value for the integral gain multiplied by the sample time.

---

**Note** PID tuning tools, such as the **PID Tuner** app and Closed-Loop PID Autotuner block, tune the gain **I** but not **I\*Ts**. Therefore, multiply the integral gain value you obtain from a tuning tool by the sample time before you write it to this parameter.

---

When you use **I\*Ts** instead of **I**, the block requires fewer calculations to perform integration. This improves the execution time of the generated code.

**Tunable:** No

#### **Dependencies**

To enable this parameter, in the **Main** tab, set the controller-parameters **Source** to `internal`, set **Controller** to a type that has integral action, and enable the **Use I\*Ts** parameter.

#### **Programmatic Use**

**Block Parameter:** I

**Type:** scalar, vector

**Default:** 1

### **Use I\*Ts — Use integral gain multiplied by sample time**

on (default) | off

For discrete-time controllers with integral action, the block takes the integral gain as an input and multiplies it by the sample time internally as a part of performing the integration. You can enable this parameter to specify integral gain multiplied by sample time as input (**I\*Ts**) in place of the integral gain (**I**). Doing so reduces the number of internal calculations and is useful when you want to improve the execution time of your generated code.

#### **Dependencies**

To enable this parameter, set **Controller** to a controller type that has integral action.

#### **Programmatic Use**

**Block Parameter:** UseKiTs

**Type:** string, character vector

**Values:** "on", "off"

**Default:** "on"

### **Derivative (D) — Derivative gain**

0 (default) | scalar | vector

Specify a finite, real gain value for the derivative gain.

**Tunable:** Yes

#### **Dependencies**

To enable this parameter, in the **Main** tab, set the controller-parameters **Source** to `internal`, and set **Controller** to `PID` or `PD`.

**Programmatic Use****Block Parameter:** D**Type:** scalar, vector**Default:** 0**Use filtered derivative – Apply filter to derivative term**

on (default) | off

For discrete-time PID controllers only, clear this option to replace the filtered derivative with an unfiltered discrete-time differentiator. When you do so, the derivative term of the controller transfer function becomes:

$$D \frac{z-1}{zT_s}$$

For continuous-time PID controllers, the derivative term is always filtered.

**Dependencies**

To enable this parameter, set **Time domain** to Discrete-time, and set **Controller** to a type that has derivative action.

**Programmatic Use****Block Parameter:** UseFilter**Type:** string, character vector**Values:** "on", "off"**Default:** "on"**Filter coefficient (N) – Derivative filter coefficient**

100 (default) | scalar | vector

Specify a finite, real gain value for the filter coefficient. The filter coefficient determines the pole location of the filter in the derivative action of the block. The location of the filter pole depends on the **Time domain** parameter.

- When **Time domain** is Continuous-time, the pole location is  $s = -N$ .
- When **Time domain** is Discrete-time, the pole location depends on the **Filter method** parameter.

Filter Method	Location of Filter Pole
Forward Euler	$z_{pole} = 1 - NT_s$
Backward Euler	$z_{pole} = \frac{1}{1 + NT_s}$
Trapezoidal	$z_{pole} = \frac{1 - NT_s/2}{1 + NT_s/2}$

The block does not support  $N = \text{Inf}$  (ideal unfiltered derivative). When the **Time domain** is Discrete-time, you can clear **Use filtered derivative** to remove the derivative filter.

**Tunable:** Yes

## Dependencies

To enable this parameter, in the **Main** tab, set the controller-parameters **Source** to **internal** and set **Controller** to **PID** or **PD**.

### Programmatic Use

**Block Parameter:** N

**Type:** scalar, vector

**Default:** 100

## Select tuning method — Tool for automatic tuning of controller coefficients

Transfer Function Based (PID Tuner App) (default) | Frequency Response Based

If you have Simulink Control Design software, you can automatically tune the PID coefficients. To do so, use this parameter to select a tuning tool, and click **Tune**.

### Transfer Function Based (PID Tuner App)

Use **PID Tuner**, which lets you interactively tune PID coefficients while examining relevant system responses to validate performance. By default, **PID Tuner** works with a linearization of your plant model. For models that cannot be linearized, you can tune PID coefficients against a plant model estimated from simulated or measured response data. For more information, see “Introduction to Model-Based PID Tuning in Simulink” (Simulink Control Design).

### Frequency Response Based

Use **Frequency Response Based PID Tuner**, which tunes PID controller coefficients based on frequency-response estimation data obtained by simulation. This tuning approach is especially useful for plants that are not linearizable or that linearize to zero. For more information, see “Design PID Controller from Plant Frequency-Response Data” (Simulink Control Design).

Both of these tuning methods assume a single-loop control configuration. Simulink Control Design software includes other tuning approaches that suit more complex configurations. For information about other ways to tune a PID Controller block, see “Choose a Control Design Approach” (Simulink Control Design).

## Enable zero-crossing detection — Detect zero crossings on reset and on entering or leaving a saturation state

on (default) | off

Zero-crossing detection can accurately locate signal discontinuities without resorting to excessively small time steps that can lead to lengthy simulation times. If you select **Limit output** or activate **External reset** in your PID Controller block, activating zero-crossing detection can reduce computation time in your simulation. Selecting this parameter activates zero-crossing detection:

- At initial-state reset
- When entering an upper or lower saturation state
- When leaving an upper or lower saturation state

For more information about zero-crossing detection, see “Zero-Crossing Detection”.

### Programmatic Use

**Block Parameter:** ZeroCross

**Type:** string, character vector

**Values:** "on", "off"

**Default:** "on"

## Initialization

### Source — Source for integrator and derivative initial conditions

`external` (default) | `internal`

Simulink uses initial conditions to initialize the integrator and derivative-filter (or the unfiltered derivative) output at the start of a simulation or at a specified trigger event. (See the **External reset** parameter.) These initial conditions determine the initial block output. Use this parameter to select how to supply the initial condition values to the block.

#### `internal`

Specify the initial conditions using the **Integrator Initial condition** and **Filter Initial condition** parameters. If **Use filtered derivative** is not selected, use the **Differentiator** parameter to specify the initial condition for the unfiltered differentiator instead of a filter initial condition.

#### `external`

Specify the initial conditions externally using block inputs. Additional input ports **I<sub>0</sub>** and **D<sub>0</sub>** appear on the block. If **Use filtered derivative** is not selected, supply the initial condition for the unfiltered differentiator at **D<sub>0</sub>** instead of a filter initial condition.

#### Programmatic Use

**Block Parameter:** `InitialConditionSource`

**Type:** string, character vector

**Values:** "internal", "external"

**Default:** "internal"

### Integrator — Integrator initial condition

0 (default) | scalar | vector

Simulink uses the integrator initial condition to initialize the integrator at the start of a simulation or at a specified trigger event (see **External reset**). The integrator initial condition and the filter initial condition determine the initial output of the PID controller block.

The integrator initial condition cannot be NaN or Inf.

#### Dependencies

To use this parameter, in the **Initialization** tab, set **Source** to `internal`, and set **Controller** to a type that has integral action.

#### Programmatic Use

**Block Parameter:** `InitialConditionForIntegrator`

**Type:** scalar, vector

**Default:** 0

### Filter — Filter initial condition

0 (default) | scalar | vector

Simulink uses the filter initial condition to initialize the derivative filter at the start of a simulation or at a specified trigger event (see **External reset**). The integrator initial condition and the filter initial condition determine the initial output of the PID controller block.

The filter initial condition cannot be NaN or Inf.

**Dependencies**

To use this parameter, in the **Initialization** tab, set **Source** to `internal`, and use a controller that has a derivative filter.

**Programmatic Use**

**Block Parameter:** `InitialConditionForFilter`

**Type:** scalar, vector

**Default:** 0

**Differentiator — Initial condition for unfiltered derivative**

0 (default) | scalar | vector

When you use an unfiltered derivative, Simulink uses this parameter to initialize the differentiator at the start of a simulation or at a specified trigger event (see **External reset**). The integrator initial condition and the derivative initial condition determine the initial output of the PID controller block.

The derivative initial condition cannot be NaN or Inf.

**Dependencies**

To use this parameter, set **Time domain** to `Discrete-time`, clear the **Use filtered derivative** check box, and in the **Initialization** tab, set **Source** to `internal`.

**Programmatic Use**

**Block Parameter:** `DifferentiatorICPrevScaledInput`

**Type:** scalar, vector

**Default:** 0

**Initial condition setting — Location at which initial condition is applied**

Auto (default) | Output

Use this parameter to specify whether to apply the **Integrator Initial condition** and **Filter Initial condition** parameter to the corresponding block state or output. You can change this parameter at the command line only, using `set_param` to set the `InitialConditionSetting` parameter of the block.

**Auto**

Use this option in all situations except when the block is in a triggered subsystem or a function-call subsystem and simplified initialization mode is enabled.

**Output**

Use this option when the block is in a triggered subsystem or a function-call subsystem and simplified initialization mode is enabled.

For more information about the **Initial condition setting** parameter, see the Discrete-Time Integrator block.

This parameter is only accessible through programmatic use.

**Programmatic Use**

**Block Parameter:** `InitialConditionSetting`

**Type:** string, character vector

**Values:** "Auto", "Output"

**Default:** "Auto"

**External reset — Trigger for resetting integrator and filter values**

level (default) | none | rising | falling | either

Specify the trigger condition that causes the block to reset the integrator and filter to initial conditions. (If **Use filtered derivative** is not selected, the trigger resets the integrator and differentiator to initial conditions.) Selecting any option other than **none** enables the **Reset** port on the block for the external reset signal.

**none**

The integrator and filter (or differentiator) outputs are set to initial conditions at the beginning of simulation, and are not reset during simulation.

**rising**

Reset the outputs when the reset signal has a rising edge.

**falling**

Reset the outputs when the reset signal has a falling edge.

**either**

Reset the outputs when the reset signal either rises or falls.

**level**

Reset the outputs when the reset signal either:

- Is nonzero at the current time step
- Changes from nonzero at the previous time step to zero at the current time step

This option holds the outputs to the initial conditions while the reset signal is nonzero.

**Dependencies**

To enable this parameter, set **Controller** to a type that has derivative or integral action.

**Programmatic Use****Block Parameter:** ExternalReset**Type:** string, character vector**Values:** "none", "rising", "falling", "either", "level"**Default:** "level"**Ignore reset when linearizing — Force linearization to ignore reset**

off (default) | on

Select to force Simulink and Simulink Control Design linearization commands to ignore any reset mechanism specified in the **External reset** parameter. Ignoring reset states allows you to linearize a model around an operating point even if that operating point causes the block to reset.

**Programmatic Use****Block Parameter:** IgnoreLimit**Type:** string, character vector**Values:** "off", "on"**Default:** "off"**Enable tracking mode — Activate signal tracking**

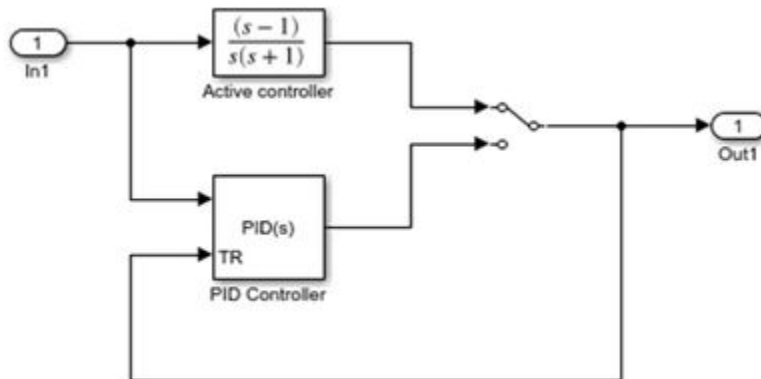
off (default) | on

Signal tracking lets the block output follow a tracking signal that you provide at the **TR** port. When signal tracking is active, the difference between the tracking signal and the block output is fed back

to the integrator input with a gain  $K_t$ , specified by the **Tracking gain (Kt)** parameter. Signal tracking has several applications, including bumpless control transfer and avoiding windup in multiloop control structures.

### Bumpless control transfer

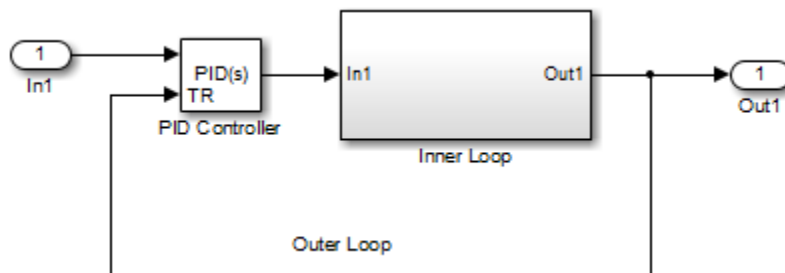
Use signal tracking to achieve bumpless control transfer in systems that switch between two controllers. Suppose you want to transfer control between a PID controller and another controller. To do so, connecting the controller output to the **TR** input as shown in the following illustration.



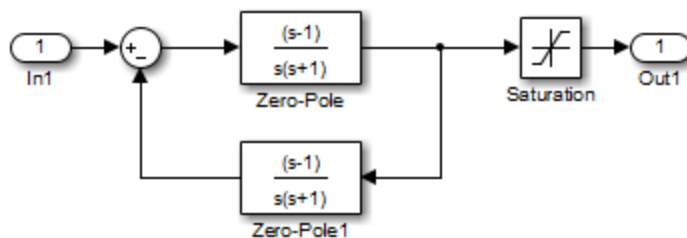
For more information, see “Bumpless Control Transfer”.

### Multiloop control

Use signal tracking to prevent block windup in multiloop control approaches, as in the following model.



The Inner Loop subsystem contains the blocks shown in the following diagram.



Because the PID controller tracks the output of the inner loop, its output never exceeds the saturated inner-loop output. For more details, see “Prevent Block Windup in Multiloop Control”.

**Dependencies**

To enable this parameter, set **Controller** to a type that has integral action.

**Programmatic Use**

**Block Parameter:** TrackingMode

**Type:** string, character vector

**Values:** "off", "on"

**Default:** "off"

**Tracking coefficient (Kt) — Gain of signal-tracking feedback loop**

1 (default) | scalar

When you select **Enable tracking mode**, the difference between the signal **TR** and the block output is fed back to the integrator input with a gain **Kt**. Use this parameter to specify the gain in that feedback loop.

**Dependencies**

To enable this parameter, select **Enable tracking mode**.

**Programmatic Use**

**Block Parameter:** Kt

**Type:** scalar

**Default:** 1

**Output Saturation****Limit Output — Limit block output to specified saturation values**

on (default) | off

Activating this option limits the block output, so that you do not need a separate Saturation block after the controller. It also allows you to activate the anti-windup mechanism built into the block (see the **Anti-windup method** parameter). Specify the output saturation limits using the **Lower limit** and **Upper limit** parameters. You can also specify the saturation limits externally as block input ports.

**Programmatic Use**

**Block Parameter:** LimitOutput

**Type:** string, character vector

**Values:** "off", "on"

**Default:** "on"

**Source — Source for output saturation limits**

internal (default) | external

Use this parameter to specify how to supply the upper and lower saturation limits of the block output.

internal

Specify the output saturation limits using the **Upper limit** and **Lower limit** parameters.

external

Specify the output saturation limits externally using block input ports. The additional input ports **up** and **lo** appear on the block. You can use the input ports to implement the upper and lower output saturation limits determined by logic or other calculations in the Simulink model and passed to the block.



**Programmatic Use****Block Parameter:** SatLimitsSource**Type:** string, character vector**Values:** "internal", "external"**Default:** "internal"**Upper limit — Upper saturation limit for block output**

1 (default) | scalar

Specify the upper limit for the block output. The block output is held at the **Upper saturation limit** whenever the weighted sum of the proportional, integral, and derivative actions exceeds that value.

**Dependencies**

To enable this parameter, select **Limit output**.

**Programmatic Use****Block Parameter:** UpperSaturationLimit**Type:** scalar**Default:** 1**Lower limit — Lower saturation limit for block output**

-1 (default) | scalar

Specify the lower limit for the block output. The block output is held at the **Lower saturation limit** whenever the weighted sum of the proportional, integral, and derivative actions goes below that value.

**Dependencies**

To enable this parameter, select **Limit output**.

**Programmatic Use****Block Parameter:** LowerSaturationLimit**Type:** scalar**Default:** -1**Ignore saturation when linearizing — Force linearization to ignore output limits**

on (default) | off

Force Simulink and Simulink Control Design linearization commands to ignore block output limits specified in the **Upper limit** and **Lower limit** parameters. Ignoring output limits allows you to linearize a model around an operating point even if that operating point causes the block to exceed the output limits.

**Dependencies**

To enable this parameter, select the **Limit output** parameter.

**Programmatic Use****Block Parameter:** LinearizeAsGain**Type:** string, character vector**Values:** "off", "on"**Default:** "on"**Anti-windup method — Integrator anti-windup method**

clamping (default) | none | back-calculation

When you select **Limit output** and the weighted sum of the controller components exceeds the specified output limits, the block output holds at the specified limit. However, the integrator output can continue to grow (integrator windup), increasing the difference between the block output and the sum of the block components. In other words, the internal signals in the block can be unbounded even if the output appears bounded by saturation limits. Without a mechanism to prevent integrator windup, two results are possible:

- If the sign of the input signal never changes, the integrator continues to integrate until it overflows. The overflow value is the maximum or minimum value for the data type of the integrator output.
- If the sign of the input signal changes once the weighted sum has grown beyond the output limits, it can take a long time to unwind the integrator and return the weighted sum within the block saturation limit.

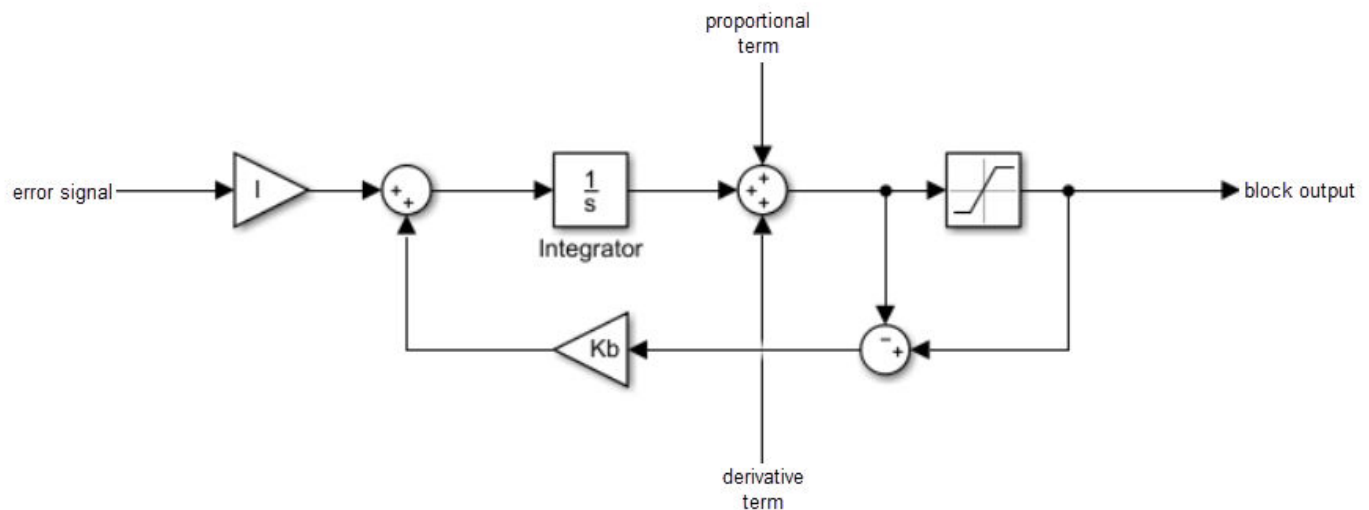
In either case, controller performance can suffer. To combat the effects of windup without an anti-windup mechanism, it may be necessary to detune the controller (for example, by reducing the controller gains), resulting in a sluggish controller. To avoid this problem, activate an anti-windup mechanism using this parameter:

none

Do not use an anti-windup mechanism.

back-calculation

Unwind the integrator when the block output saturates by feeding back to the integrator the difference between the saturated and unsaturated control signal. The following diagram represents the back-calculation feedback circuit for a continuous-time controller. To see the actual feedback circuit for your controller configuration, right-click on the block and select **Mask > Look Under Mask**.



Use the **Back-calculation coefficient (Kb)** parameter to specify the gain of the anti-windup feedback circuit. It is usually satisfactory to set  $K_b = I$ , or for controllers with derivative action,  $K_b = \sqrt{I \cdot D}$ . Back-calculation can be effective for plants with relatively large dead time [1].

clamping

Integration stops when the sum of the block components exceeds the output limits and the integrator output and block input have the same sign. Integration resumes when the sum of the

block components exceeds the output limits and the integrator output and block input have opposite sign. Clamping is sometimes referred to as conditional integration.

Clamping can be useful for plants with relatively small dead times, but can yield a poor transient response for large dead times [1].

### Dependencies

To enable this parameter, select the **Limit output** parameter.

### Programmatic Use

**Block Parameter:** AntiWindupMode

**Type:** string, character vector

**Values:** "none", "back-calculation", "clamping"

**Default:** "clamping"

### Back-calculation coefficient (Kb) — Gain coefficient of anti-windup feedback loop

1 (default) | scalar

The **back-calculation** anti-windup method unwinds the integrator when the block output saturates. It does so by feeding back to the integrator the difference between the saturated and unsaturated control signal. Use the **Back-calculation coefficient (Kb)** parameter to specify the gain of the anti-windup feedback circuit. For more information, see the **Anti-windup method** parameter.

### Dependencies

To enable this parameter, select the **Limit output** parameter, and set the **Anti-windup method** parameter to **back-calculation**.

### Programmatic Use

**Block Parameter:** Kb

**Type:** scalar

**Default:** 1


### Data Types

The parameters in this tab are primarily of use in fixed-point code generation using Fixed-Point Designer™. They define how numeric quantities associated with the block are stored and processed when you generate code.

If you need to configure data types for fixed-point code generation, click **Open Fixed-Point Tool** and use that tool to configure the rest of the parameters in the tab. For information about using Fixed-Point Tool, see “Autoscaling Data Objects Using the Fixed-Point Tool” (Fixed-Point Designer).

After you use Fixed-Point Tool, you can use the parameters in this tab to make adjustments to fixed-point data-type settings if necessary. For each quantity associated with the block, you can specify:

- Floating-point or fixed-point data type, including whether the data type is inherited from upstream values in the block.
- The minimum and maximum values for the quantity, which determine how the quantity is scaled for fixed-point representation.

For assistance in selecting appropriate values, click  to open the Data Type Assistant for the corresponding quantity. For more information, see “Specify Data Types Using Data Type Assistant”.

Main	Initialization	Output Saturation	Data Types	State Attributes
Fixed-point operational parameters				
Integer rounding mode: <span>Simplest</span>				
<input type="checkbox"/> Saturate on integer overflow				
<input type="checkbox"/> Lock data type settings against changes by the fixed-point tools <span>Open Fixed-Point Tool...</span>				
Data Type		Minimum		Maximum
P product output:	<span>Inherit: Same as first input</span>	<span>&gt;&gt;</span>	<span>[]</span>	<span>[]</span>
I product output:	<span>Inherit: Same as first input</span>	<span>&gt;&gt;</span>	<span>[]</span>	<span>[]</span>
Sum output:	<span>Inherit: Same as first input</span>	<span>&gt;&gt;</span>	<span>[]</span>	<span>[]</span>
Saturation output:	<span>Inherit: Inherit via back propagation</span>	<span>&gt;&gt;</span>	<span>[]</span>	<span>[]</span>
▶ Additional data types				

The specific quantities listed in the Data Types tab vary depending on how you configure the PID controller block. In general, you can configure data types for the following types of quantities:

- Product output — Stores the result of a multiplication carried out under the block mask. For example, **P product output** stores the output of the gain block that multiplies the block input with the proportional gain **P**.
- Parameter — Stores the value of a numeric block parameter, such as **P**, **I**, or **D**.
- Block output — Stores the output of a block that resides under the PID controller block mask. For example, use **Integrator output** to specify the data type of the output of the block called Integrator. This block resides under the mask in the Integrator subsystem, and computes integrator term of the controller action.
- Accumulator — Stores values associated with a sum block. For example, **SumI2 Accumulator** sets the data type of the accumulator associated with the sum block SumI2. This block resides under the mask in the Back Calculation subsystem of the Anti-Windup subsystem.

In general, you can find the block associated with any listed parameter by looking under the PID Controller block mask and examining its subsystems. You can also use the Model Explorer to search under the mask for the listed parameter name, such as SumI2. (See **Model Explorer**.)

### Matching Input and Internal Data Types

By default, the data types for product output and block output are set to **Inherit: Same as first input**. With this setting, the block uses the data type of its first input signal for these signals.

When data type is set to **Inherit: Inherit via internal** rule, Simulink chooses data types to balance numerical accuracy, performance, and generated code size, while accounting for the properties of the embedded target hardware.

Under some conditions, incompatibility can occur between data types within the block. For instance, in continuous time, the Integrator block under the mask can accept only signals of type **double**. If the block input signal is a type that cannot be converted to **double**, such as **uint16**, the internal rules for type inheritance generate an error when you generate code.

To avoid such errors, you can use the Data Types settings to force a data type conversion. For instance, you can explicitly set **P product output**, **I product output**, and **D product output** to `double`, ensuring that the signals reaching the continuous-time integrators are of type `double`.

In general, it is not recommended to use the block in continuous time for code generation applications. However, similar data type errors can occur in discrete time, if you explicitly set some values to data types that are incompatible with downstream signal constraints within the block. In such cases, use the Data Types settings to ensure that all data types are internally compatible.

### Fixed-Point Operational Parameters

#### Integer rounding mode — Rounding mode for fixed-point operations

`Simplest` (default) | `Floor` | `Ceiling` | `Convergent` | `Nearest` | `Round` | `Zero`

Specify the rounding mode for fixed-point operations. For more information, see “Rounding” (Fixed-Point Designer).

Block parameters always round to the nearest representable value. To control the rounding of a block parameter, enter an expression using a MATLAB® rounding function into the mask field.

#### Programmatic Use

**Block Parameter:** `RndMeth`

**Type:** character vector

**Values:** `'Ceiling'` | `'Convergent'` | `'Floor'` | `'Nearest'` | `'Round'` | `'Simplest'` | `'Zero'`

**Default:** `'Simplest'`

#### Saturate on integer overflow — Method of overflow action

`off` (default) | `on`

Specify whether overflows saturate or wrap.

- `off` — Overflows wrap to the appropriate value that the data type can represent.

For example, the number 130 does not fit in a signed 8-bit integer and wraps to -126.

- `on` — Overflows saturate to either the minimum or maximum value that the data type can represent.

For example, an overflow associated with a signed 8-bit integer can saturate to -128 or 127.

---

### Tip

- Consider selecting this check box when your model has a possible overflow and you want explicit saturation protection in the generated code.
- Consider clearing this check box when you want to optimize efficiency of your generated code.

Clearing this check box also helps you to avoid overspecifying how a block handles out-of-range signals. For more information, see “Troubleshoot Signal Range Errors”.

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
  - In general, the code generation process can detect when overflow is not possible. In this case, the code generator does not produce saturation code.
-

**Programmatic Use****Block Parameter:** SaturateOnIntegerOverflow**Type:** character vector**Values:** 'off' | 'on'**Default:** 'off'**Lock data type settings against changes by the fixed-point tools — Prevent fixed-point tools from overriding data types**

on (default) | off

Select this parameter to prevent the fixed-point tools from overriding the data types you specify on this block. For more information, see “Lock the Output Data Type Setting” (Fixed-Point Designer).

**Programmatic Use****Block Parameter:** LockScale**Type:** character vector**Values:** 'off' | 'on'**Default:** 'on'**State Attributes**

The parameters in this tab are primarily of use in code generation.

**State name (e.g., 'position') — Name for continuous-time filter and integrator states**

' ' (default) | character vector

Assign a unique name to the state associated with the integrator or the filter, for continuous-time PID controllers. (For information about state names in a discrete-time PID controller, see the **State name** parameter.) The state name is used, for example:

- For the corresponding variable in generated code
- As part of the storage name when logging states during simulation
- For the corresponding state in a linear model obtain by linearizing the block

A valid state name begins with an alphabetic or underscore character, followed by alphanumeric or underscore characters.

**Dependencies**

To enable this parameter, set **Time domain** to Continuous-time.

**Programmatic Use****Parameter:** IntegratorContinuousStateAttributes, FilterContinuousStateAttributes**Type:** character vector**Default:** ' '**State name — Names for discrete-time filter and integrator states**

empty string (default) | string | character vector

Assign a unique name to the state associated with the integrator or the filter, for discrete-time PID controllers. (For information about state names in a continuous-time PID controller, see the **State name (e.g., 'position')** parameter.)

A valid state name begins with an alphabetic or underscore character, followed by alphanumeric or underscore characters. The state name is used, for example:

- For the corresponding variable in generated code
- As part of the storage name when logging states during simulation
- For the corresponding state in a linear model obtain by linearizing the block

For more information about the use of state names in code generation, see “C Code Generation Configuration for Model Interface Elements” (Simulink Coder).

#### Dependencies

To enable this parameter, set **Time domain** to Discrete-time.

#### Programmatic Use

**Parameter:** IntegratorStateIdentifier, FilterStateIdentifier

**Type:** string, character vector

**Default:** ""

#### State name must resolve to Simulink signal object – Require that state name resolve to a signal object

off (default) | on

Select this parameter to require that the discrete-time integrator or filter state name resolves to a Simulink signal object.

#### Dependencies

To enable this parameter for the discrete-time integrator or filter state:

- 1 Set **Time domain** to Discrete-time.
- 2 Specify a value for the integrator or filter **State name**.
- 3 Set the model configuration parameter **Signal resolution** to a value other than None.

#### Programmatic Use

**Block Parameter:** IntegratorStateMustResolveToSignalObject, FilterStateMustResolveToSignalObject

**Type:** string, character vector

**Values:** "off", "on"

**Default:** "off"

## Block Characteristics

<b>Data Types</b>	double   fixed point   integer   single
<b>Direct Feedthrough</b>	no
<b>Multidimensional Signals</b>	no
<b>Variable-Size Signals</b>	no
<b>Zero-Crossing Detection</b>	no

## References

- [1] Visioli, A., "Modified Anti-Windup Scheme for PID Controllers," *IEE Proceedings - Control Theory and Applications*, Vol. 150, Number 1, January 2003

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

For discrete-time PID controllers (**Time domain** set to **Discrete-time**):

- Depends on absolute time when placed inside a triggered subsystem hierarchy.
- Generated code relies on memcpy or memset functions (string.h) under certain conditions.

For continuous-time PID controllers (**Time domain** set to **Continuous-time**):

- Consider using “Model Discretizer” to map continuous-time blocks to discrete equivalents that support code generation. To access Model Discretizer, from your model, in the **Apps** tab, under **Control Systems**, click **Model Discretizer**.
- Not recommended for production code.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

HDL Coder™ provides additional configuration options that affect HDL implementation and synthesized logic.

#### HDL Architecture

This block has a single, default HDL architecture.

#### HDL Block Properties

<b>ConstrainedOutputPipeline</b>	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
<b>InputPipeline</b>	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
<b>OutputPipeline</b>	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).

#### Restrictions

- HDL code generation is supported for discrete-time PID controllers only (**Time domain** set to **Discrete-time**).
- If the **Integrator method** is set to **BackwardEuler** or **Trapezoidal**, you cannot generate HDL code for the block under either of the following conditions:
  - **Limit output** is selected and the **Anti-Windup Method** is anything other than none.
  - **Enable tracking mode** is selected.
- To generate HDL code:



- Use a discrete-time PID controller. On the **Time domain** section, specify **Discrete-time**.
- Leave the **Use filtered derivative** check box selected.
- Specify the initial conditions of the filter and integrator internally. On the **Initialization** tab, specify **Source** as `internal`.

You can specify the filter coefficients internally and externally for HDL code generation. On the **Main** tab, for **Source**, you can use `internal` or `external`.

- Set **External reset** to `none`.
- When you use double inputs, do not set **Anti-windup Method** to `clamping`.

### **PLC Code Generation**

Generate Structured Text code using Simulink® PLC Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

Fixed-point code generation is supported for discrete-time PID controllers only (**Time domain** set to `Discrete-time`).

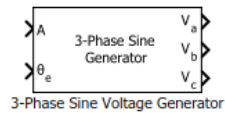
## **See Also**

**Introduced in R2022a**

## 3-Phase Sine Voltage Generator

Generate balanced three-phase sinusoidal signals

**Library:** Motor Control Blockset / Controls / Math Transforms

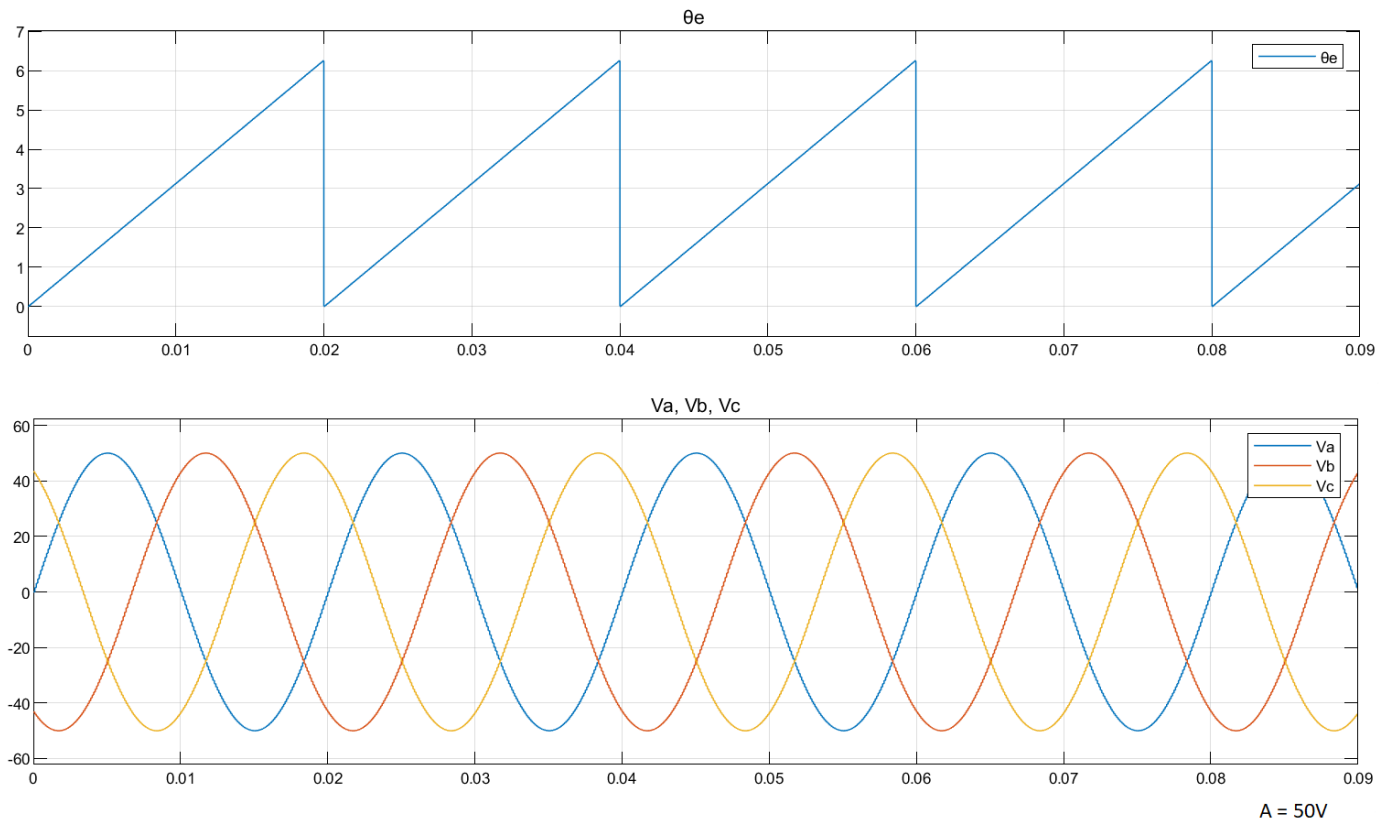


### Description

The 3-Phase Sine Voltage Generator block generates balanced, three-phase sinusoidal signals using signal amplitude and position inputs.

The block uses the lookup table approach. This approach results in optimized code-execution when used with the model settings and configuration adopted by the examples shipped in Motor Control Blockset. You can specify the number of lookup table points in the **Number of data points for lookup table** parameter.

The following image shows a plot of position input and three-phase sinusoidal output signals against time.



## Equations

The following equations describe how the block computes balanced, three-phase sinusoidal signals.

- $V_a = A \times \sin\omega t$
- $V_b = A \times \sin\left(\omega t - \frac{2\pi}{3}\right)$
- $V_c = A \times \sin\left(\omega t - \frac{4\pi}{3}\right)$

where:

- $A$  is the reference voltage amplitude (volts).
- $\omega$  is the frequency of the reference voltage position input signal ( $\theta_e$ ) (radians/ sec).
- $t$  is the time (seconds).

## Ports

### Input

#### **A — Reference voltage amplitude**

scalar

Maximum amplitude of the reference voltage signal.

Data Types: `single` | `double` | `fixed point`

#### **$\theta_e$ — Reference voltage position**

scalar

Position or phase value of the reference voltage signal.

Data Types: `single` | `double` | `fixed point`

### Output

#### **$V_a$ — a-axis component of balanced, three-phase voltage**

scalar

Balanced, three-phase voltage signal component along the  $a$ -axis of the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$V_b$ — b-axis component of balanced, three-phase voltage**

scalar

Balanced, three-phase voltage signal component along the  $b$ -axis of the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$V_c$ — c-axis component of balanced, three-phase voltage**

scalar

Balanced, three-phase voltage signal component along the  $c$ -axis of the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

## Parameters

### Theta units — Unit of $\theta_e$

Per-unit (default) | Radians | Degrees

Unit of the reference voltage position that you provide as input.

### Number of data points for lookup table — Size of lookup table

1024 (default) | scalar

Size of the lookup table.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Position Generator

### Topics

“Open-Loop and Closed-Loop Control”

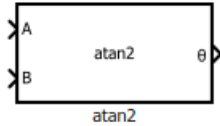
“Field-Oriented Control (FOC)”

### Introduced in R2020a

# atan2

Compute four-quadrant arctangent

**Library:** Motor Control Blockset / Controls / Math Transforms



## Description

The atan2 block performs the four-quadrant arctangent on two real numbers.

## Equations

This equation describes how the block computes the four-quadrant arctangent ( $\theta$ ).

$$\text{Theta} = \text{atan2}(A, B) = \begin{cases} \arctan\left(\frac{A}{B}\right) & \text{if } B > 0, \\ \arctan\left(\frac{A}{B}\right) + \pi & \text{if } B < 0 \text{ and } A \geq 0, \\ \arctan\left(\frac{A}{B}\right) - \pi & \text{if } B < 0 \text{ and } A < 0, \\ +\frac{\pi}{2} & \text{if } B = 0 \text{ and } A > 0, \\ -\frac{\pi}{2} & \text{if } B = 0 \text{ and } A < 0, \\ \text{undefined} & \text{if } B = 0 \text{ and } A = 0. \end{cases}$$

where:

$$-\pi < \text{Theta} \leq \pi \quad (\text{Radians})$$

## Ports

### Input

#### A — y-coordinate value (real number)

scalar

Real number on the y-axis that you provide as input to the block.

Data Types: single | double | fixed point

#### B — x-coordinate value (real number)

scalar

Real number on the x-axis that you provide as input to the block.

Data Types: single | double | fixed point

## Output

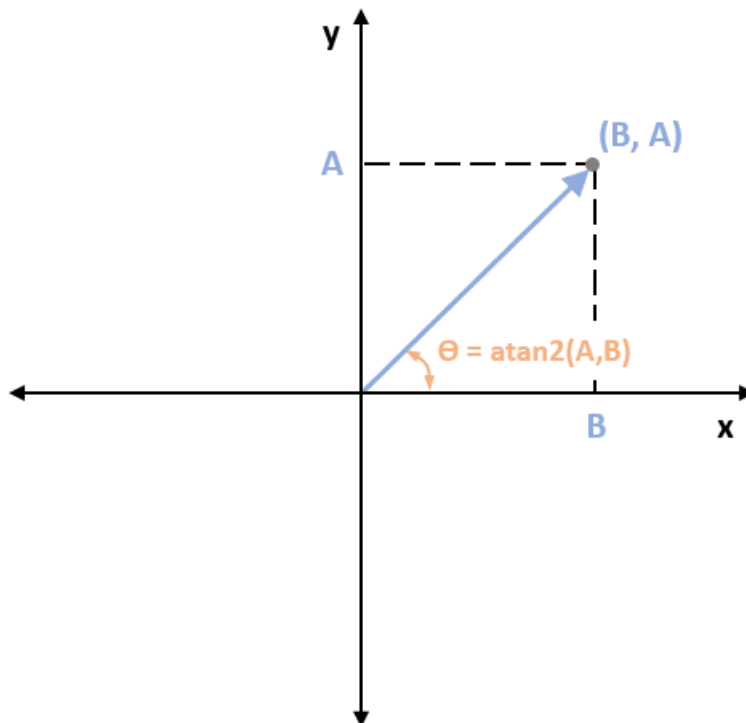
### $\theta$ – Angle represented by arctangent

scalar

Angle represented by arctangent. This is the angle made by a vector from the origin to a specified point (x,y) with the positive x-axis.

Data Types: `single` | `double` | `fixed point`

The following figure shows the representation of input values A, B, and arctangent on the x-y coordinate plane.



## Parameters

### Output unit – Unit of output values

Radians (default) | PerUnit

Unit of the output values.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

**Introduced in R2020a**

## Clarke Transform

Implement  $ab$  to  $\alpha\beta$  transformation

**Library:** Motor Control Blockset / Controls / Math Transforms



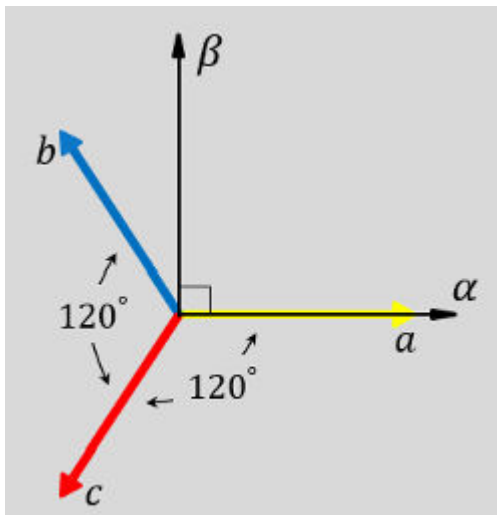
### Description

The Clarke Transform block computes the Clarke transformation of balanced three-phase components in the  $abc$  reference frame and outputs the balanced two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame.

The block accepts two signals out of the three phases ( $abc$ ), automatically calculates the third signal, and outputs the corresponding components in the  $\alpha\beta$  reference frame.

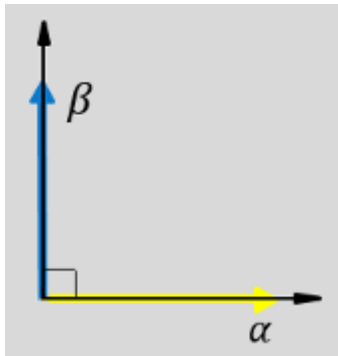
For example, the block accepts  $a$  and  $b$  input values where the phase- $a$  axis aligns with the  $\alpha$ -axis.

- This figure shows the direction of the magnetic axes of the stator windings in the  $abc$  reference frame and the stationary  $\alpha\beta$  reference frame.

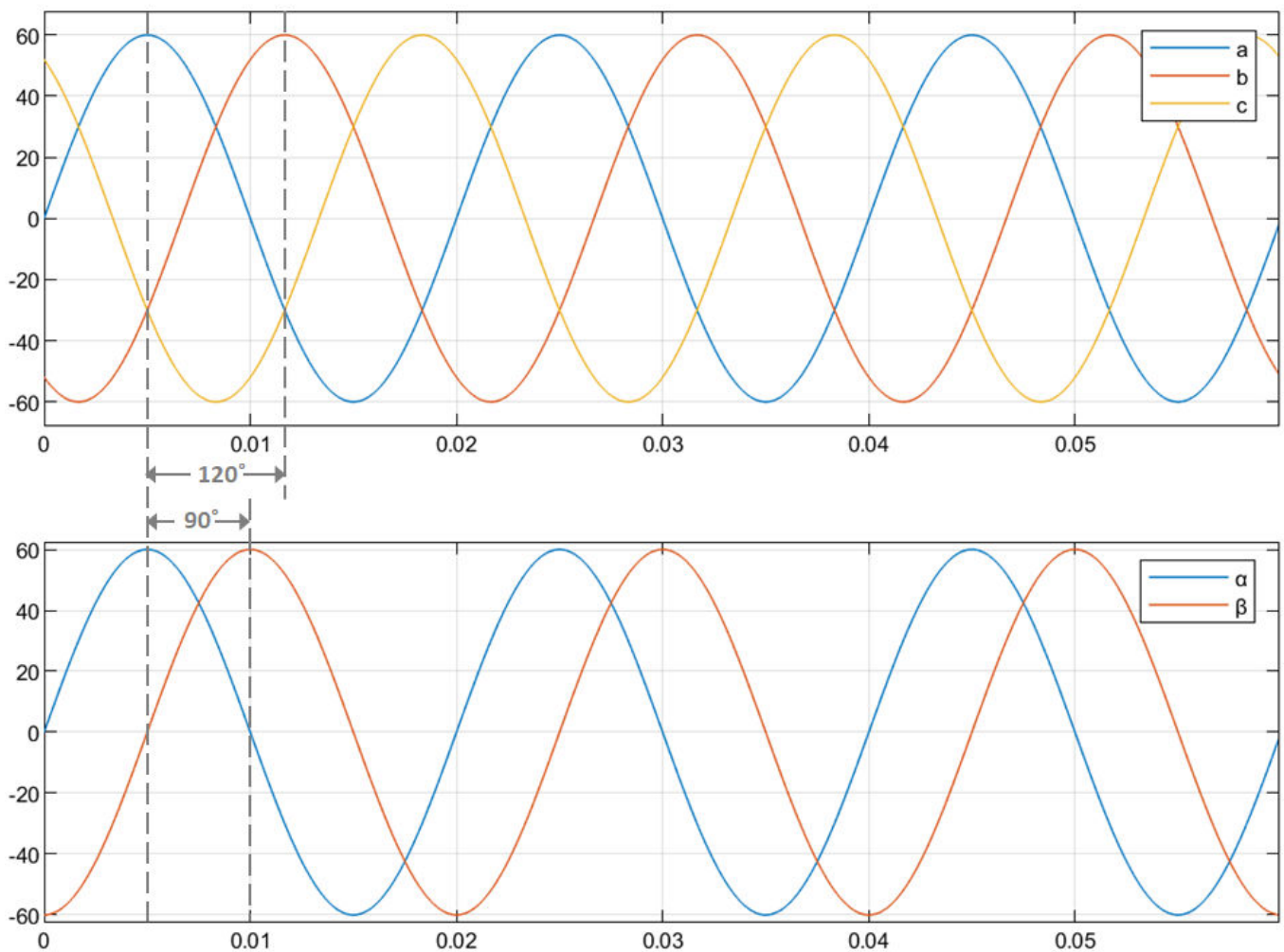


- This figure shows the equivalent  $\alpha$  and  $\beta$  components in the stationary  $\alpha\beta$  reference frame.





- The time-response of the individual components of equivalent balanced  $abc$  and  $\alpha\beta$  systems.



### Equations

The following equation describes the Clarke transform computation:

$$\begin{bmatrix} f_\alpha \\ f_\beta \\ f_0 \end{bmatrix} = \left(\frac{2}{3}\right) \times \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix}$$

For balanced systems like motors, the zero sequence component calculation is always zero. For example, the currents of the motor can be represented as,

$$i_a + i_b + i_c = 0$$

Therefore, you can use only two current sensors in three-phase motor drives, where you can calculate the third phase as,

$$i_c = -(i_a + i_b)$$

By using these equations, the block implements the Clarke transform as,

$$\begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} f_a \\ f_b \end{bmatrix}$$

where:

- $f_a$ ,  $f_b$ , and  $f_c$  are the balanced three-phase components in the  $abc$  reference frame.
- $f_\alpha$  and  $f_\beta$  are the balanced two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame.
- $f_0$  is the zero component in the stationary  $\alpha\beta$  reference frame.

## Ports

### Input

#### **a** — Phase component

scalar

Component of the three-phase system in the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **b** — Phase component

scalar

Component of the three-phase system in the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

### Output

#### **$\alpha$** — Axis component

scalar

Alpha-axis component,  $\alpha$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

**$\beta$  – Axis component**

scalar

Beta-axis component,  $\beta$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**HDL Code Generation**

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

HDL Coder provides additional configuration options that affect HDL implementation and synthesized logic.

**HDL Architecture**

This block has a single, default HDL architecture.

**HDL Block Properties**

<b>ConstrainedOutputPipeline</b>	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
<b>InputPipeline</b>	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
<b>OutputPipeline</b>	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

Inverse Clarke Transform | Park Transform

**Topics**

“Open-Loop and Closed-Loop Control”

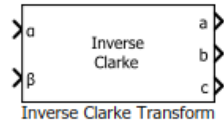
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

# Inverse Clarke Transform

Implement  $\alpha\beta$  to  $abc$  transformation

**Library:** Motor Control Blockset / Controls / Math Transforms

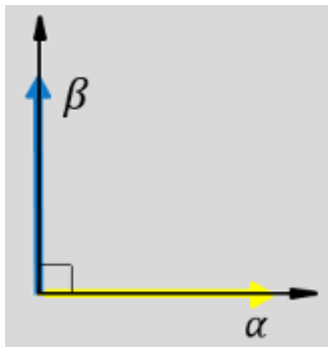


## Description

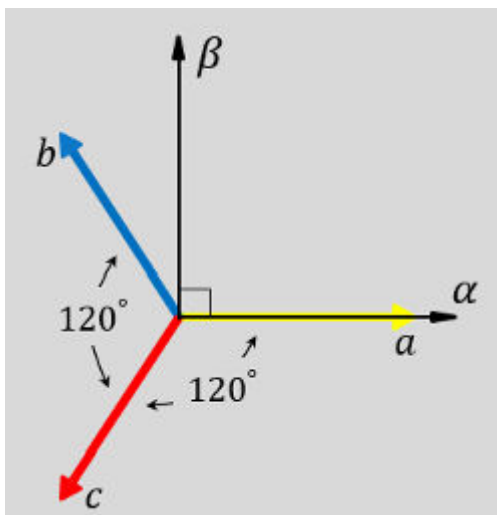
The Inverse Clarke Transform block computes the Inverse Clarke transformation of balanced, two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame. It outputs the balanced, three-phase components in the stationary  $abc$  reference frame.

The block accepts the  $\alpha$ - $\beta$  axis components as inputs and outputs the corresponding three-phase signals, where the phase- $a$  axis aligns with the  $\alpha$ -axis.

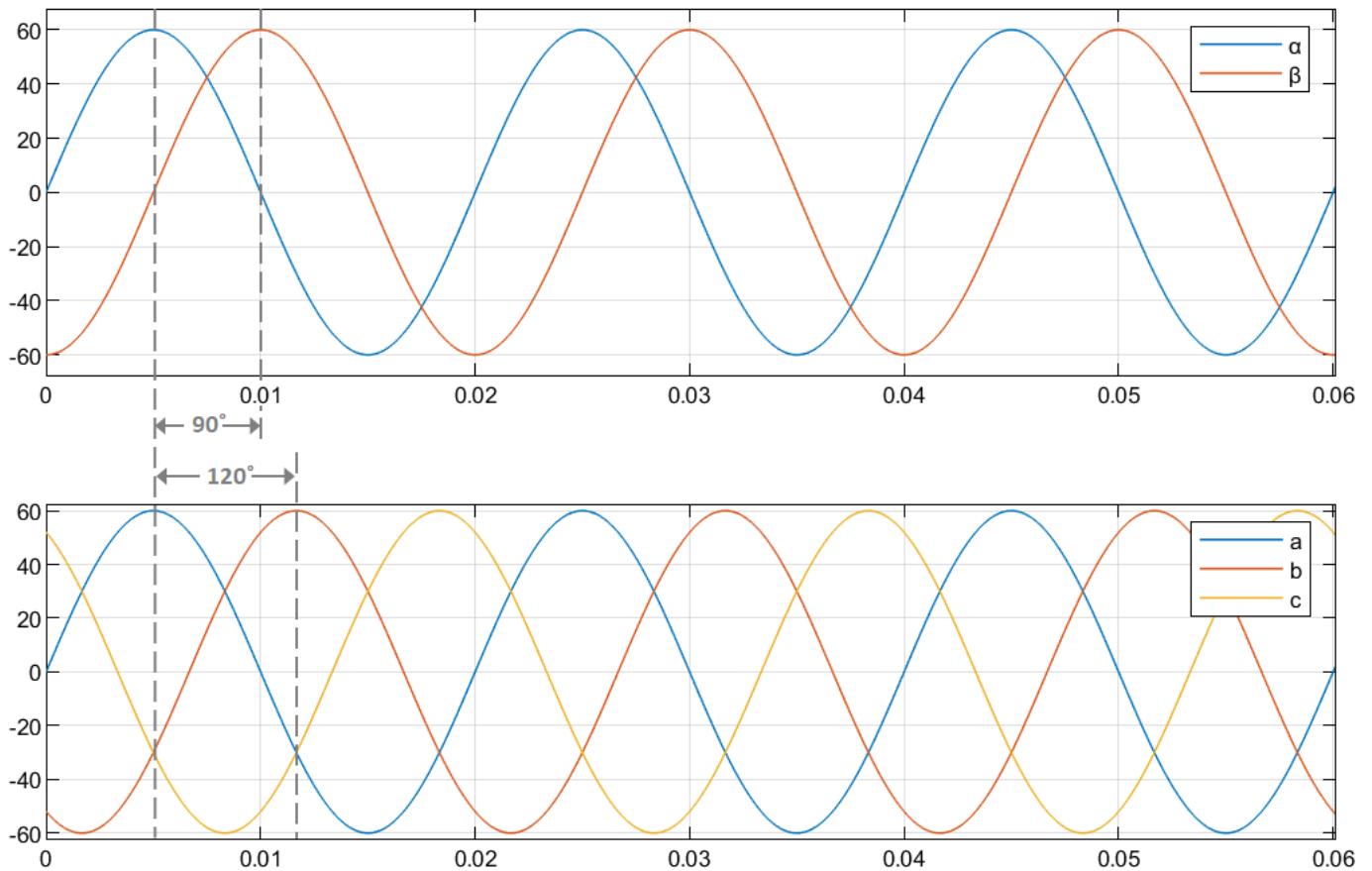
- The  $\alpha$  and  $\beta$  input components in the  $\alpha\beta$  reference frame.



- The direction of the equivalent  $a$ ,  $b$ , and  $c$  output components in the  $abc$  reference frame and the  $\alpha\beta$  reference frame.



- The time-response of the individual components of equivalent balanced  $\alpha\beta$  and  $abc$  systems.



## Equations

The following equation describes the Inverse Clarke transform computation:

$$\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \begin{bmatrix} f_\alpha \\ f_\beta \\ f_0 \end{bmatrix}$$

For balanced systems like motors, the zero sequence component calculation is always zero:

$$i_a + i_b + i_c = 0$$

Therefore, you can use only two current sensors in three-phase motor drives, where you can calculate the third phase as,

$$i_c = -(i_a + i_b)$$

By using these equations, the block implements the Inverse Clarke transform as,

$$\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix}$$

where:

- $f_\alpha$  and  $f_\beta$  are the balanced two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame.
- $f_0$  is the zero component in the stationary  $\alpha\beta$  reference frame.
- $f_a$ ,  $f_b$ , and  $f_c$  are the balanced three-phase components in the  $abc$  reference frame.

## Ports

### Input

#### $\alpha$ — Axis component

scalar

Alpha-axis component,  $\alpha$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### $\beta$ — Axis component

scalar

Beta-axis component,  $\beta$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

### Output

#### **a** — Phase component

scalar

Component of the three-phase system in the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **b** — Phase component

scalar

Component of the three-phase system in the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **c** — Phase component

scalar

Component of the three-phase system in the  $abc$  reference frame.

Data Types: `single` | `double` | `fixed point`

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

HDL Coder provides additional configuration options that affect HDL implementation and synthesized logic.

### HDL Architecture

This block has a single, default HDL architecture.

### HDL Block Properties

<b>ConstrainedOutputPipeline</b>	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
<b>InputPipeline</b>	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
<b>OutputPipeline</b>	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

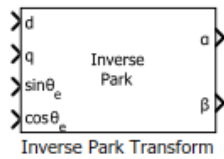
Clarke Transform

### Introduced in R2020a

## Inverse Park Transform

Implement  $dq$  to  $\alpha\beta$  transformation

**Library:** Motor Control Blockset / Controls / Math Transforms



### Description

The Inverse Park Transform block computes the inverse Park transformation of the orthogonal direct and quadrature axes components in the rotating  $dq$  reference frame. You can configure the block to align either the  $d$ - or  $q$ -axis with the  $\alpha$ -axis at time  $t = 0$ .

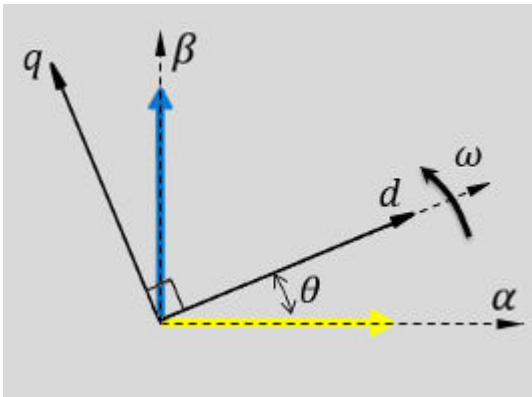
The block accepts the following inputs:

- $d$ - $q$  axes components in the rotating reference frame.
- Sine and cosine values of the corresponding angles of transformation.

It outputs the two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame.

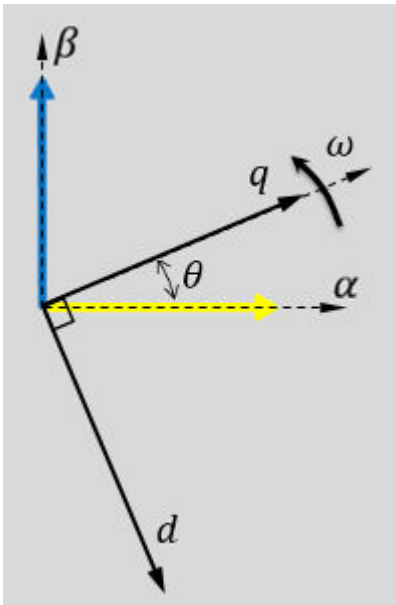
The figures show a rotating  $dq$  reference frame and the  $\alpha$ - $\beta$  axes components in an  $\alpha\beta$  reference frame for when:

- The  $d$ -axis aligns with the  $\alpha$ -axis.



- The  $q$ -axis aligns with the  $\alpha$ -axis.



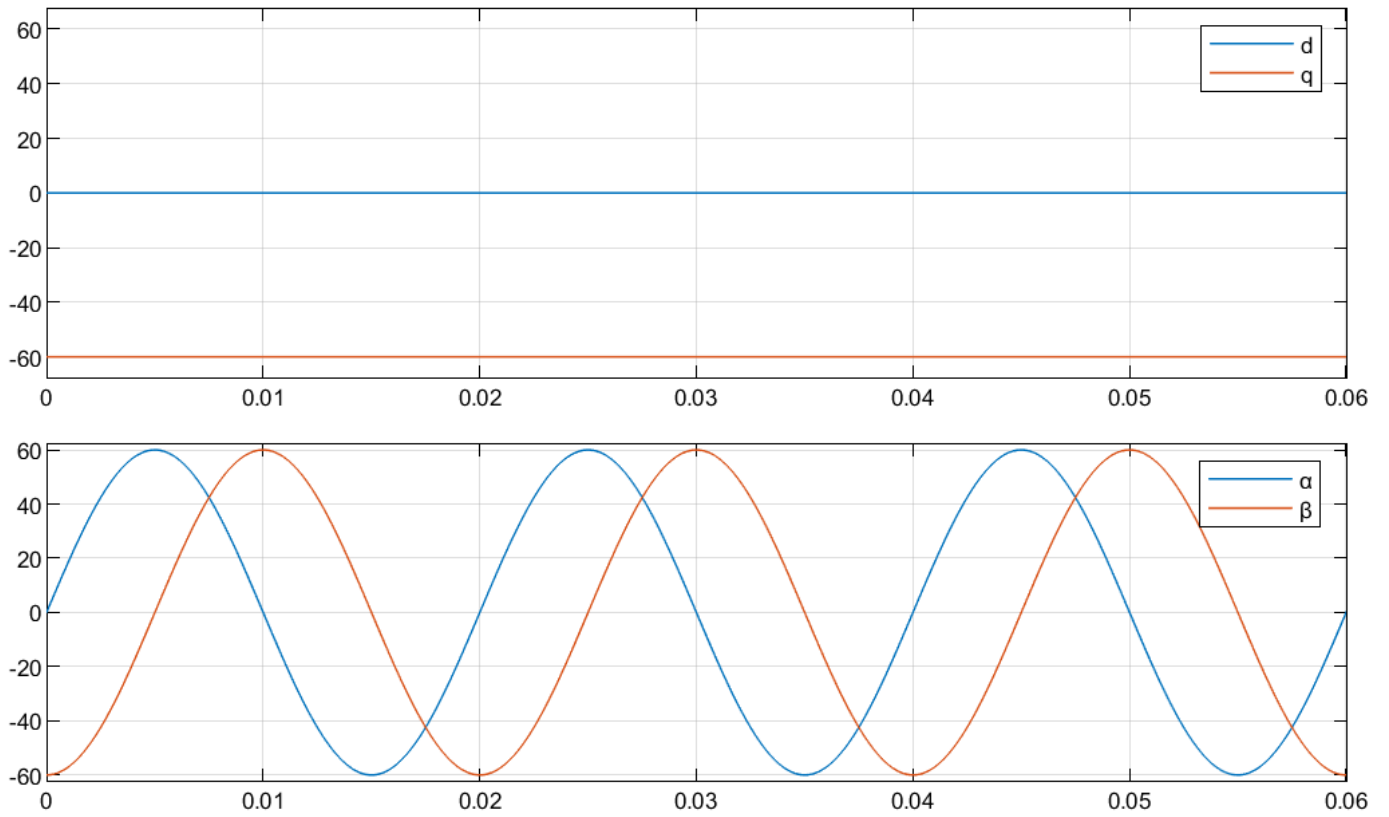


In both cases, the angle  $\theta = \omega t$ , where:

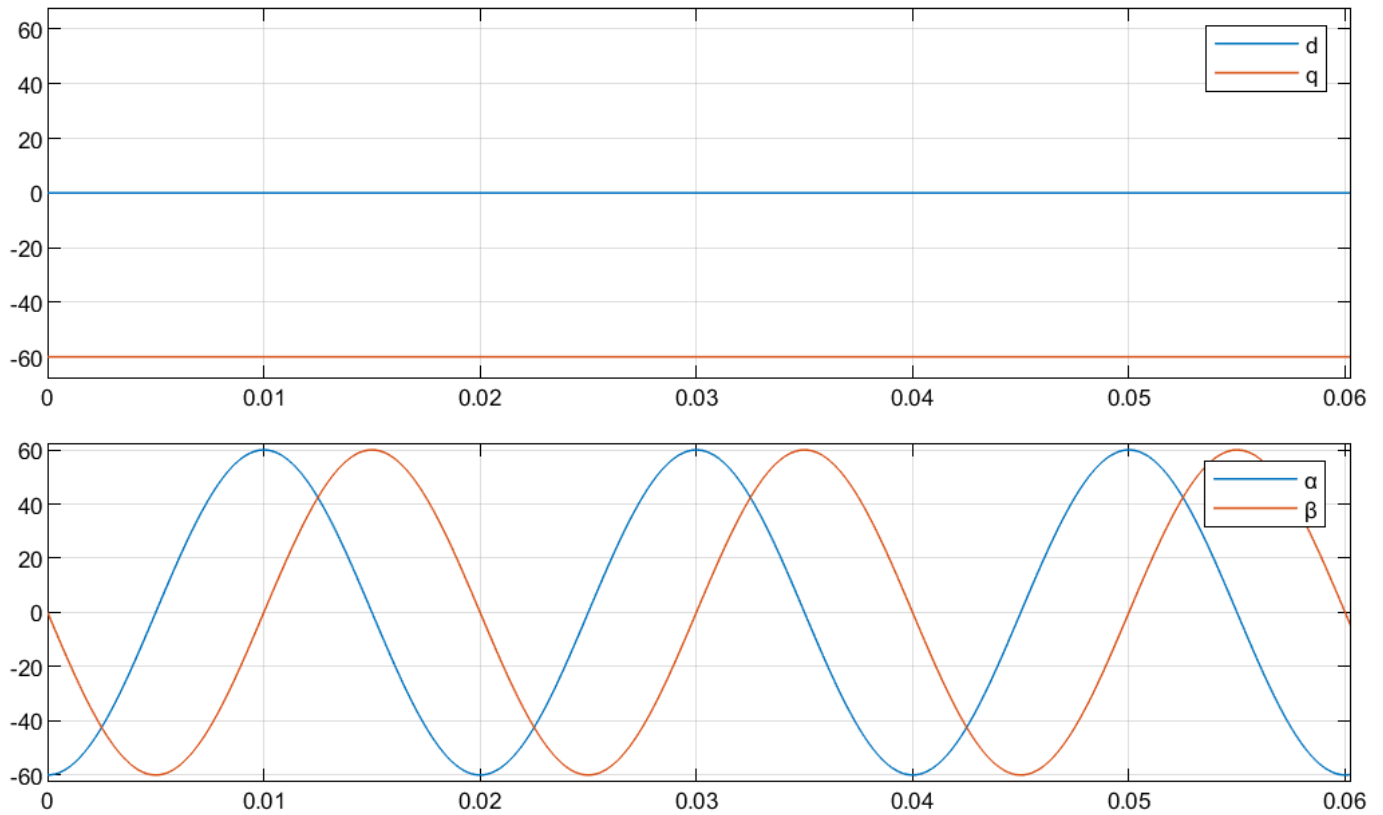
- $\theta$  is the angle between the  $\alpha$ - and  $d$ -axes for the  $d$ -axis alignment or the angle between the  $\alpha$ - and  $q$ -axes for the  $q$ -axis alignment. It indicates the angular position of the rotating  $dq$  reference frame with respect to the  $\alpha$ -axis.
- $\omega$  is the rotational speed of the  $d$ - $q$  reference frame.
- $t$  is the time, in seconds, from the initial alignment.

The figures show the time-response of the individual components of the  $\alpha\beta$  and  $dq$  reference frames when:

- The  $d$ -axis aligns with the  $\alpha$ -axis.



- The  $q$ -axis aligns with the  $\alpha$ -axis.



### Equations

The following equations describe how the block implements inverse Park transformation.

- When the  $d$ -axis aligns with the  $\alpha$ -axis.

$$\begin{bmatrix} f_{\alpha} \\ f_{\beta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} f_d \\ f_q \end{bmatrix}$$

- When the  $q$ -axis aligns with the  $\alpha$ -axis.

$$\begin{bmatrix} f_{\alpha} \\ f_{\beta} \end{bmatrix} = \begin{bmatrix} \sin\theta & \cos\theta \\ -\cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} f_d \\ f_q \end{bmatrix}$$

where:

- $f_d$  and  $f_q$  are the direct and quadrature axis orthogonal components in the rotating  $dq$  reference frame.
- $f_{\alpha}$  and  $f_{\beta}$  are the two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame.

## Ports

### Input

#### **d – Axis component**

scalar

Direct axis component,  $d$ , in the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **q – Axis component**

scalar

Quadrature axis component,  $q$ , in the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **sin $\theta_e$ – Sine value of rotational angle**

scalar

Sine value of the angle of transformation,  $\theta_e$ .  $\theta_e$  is the angle between the rotating reference frame and the  $\alpha$ -axis.

Data Types: `single` | `double` | `fixed point`

#### **cos $\theta_e$ – Cosine value of rotational angle**

scalar

Cosine value of the angle of transformation,  $\theta_e$ .  $\theta_e$  is the angle between the rotating reference frame and the  $\alpha$ -axis.

Data Types: `single` | `double` | `fixed point`

### Output

#### **$\alpha$ – Axis component**

scalar

Alpha-axis component,  $\alpha$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$\beta$ – Axis component**

scalar

Beta-axis component,  $\beta$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

## Parameters

#### **Alpha (phase-a) axis alignment – $dq$ reference frame alignment**

D-axis (default) | Q-axis

Align either the  $d$ - or  $q$ -axis of the rotating reference frame to the  $\alpha$ -axis of the stationary reference frame.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

HDL Coder provides additional configuration options that affect HDL implementation and synthesized logic.

### HDL Architecture

This block has a single, default HDL architecture.

### HDL Block Properties

<b>ConstrainedOutputPipeline</b>	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
<b>InputPipeline</b>	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
<b>OutputPipeline</b>	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Park Transform | Discrete PI Controller with anti-windup and reset | DQ Limiter | ACIM Feed Forward Control | Space Vector Generator | Sine-Cosine Lookup | PMSM Feed Forward Control

### Topics

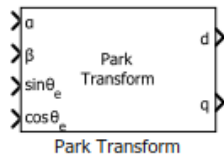
“Open-Loop and Closed-Loop Control”  
 “Field-Oriented Control (FOC)”

### Introduced in R2020a

# Park Transform

Implement  $\alpha\beta$  to  $dq$  transformation

**Library:** Motor Control Blockset / Controls / Math Transforms



## Description

The Park Transform block computes the Park transformation of two-phase orthogonal components in a stationary  $\alpha\beta$  reference frame.

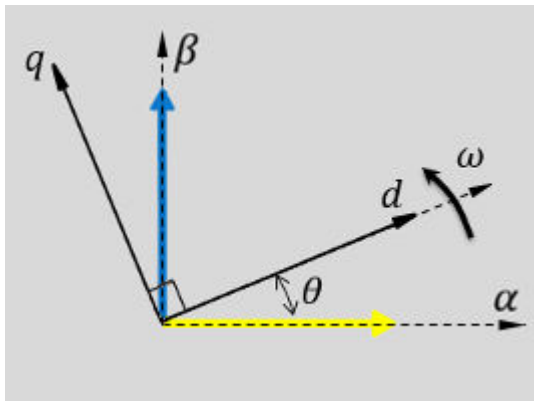
The block accepts the following inputs:

- $\alpha$ - $\beta$  axes components in the stationary reference frame.
- Sine and cosine values of the corresponding angles of transformation.

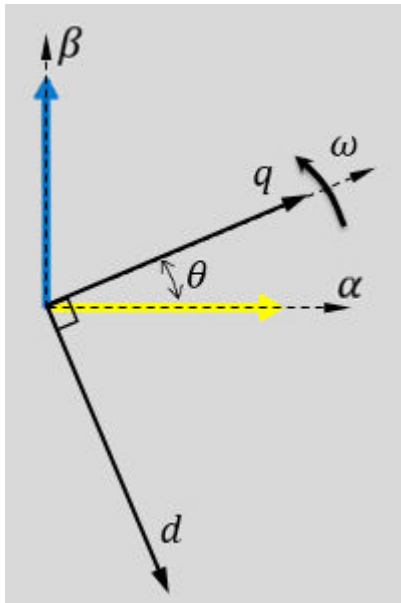
It outputs orthogonal direct and quadrature axis components in the rotating  $dq$  reference frame. You can configure the block to align either the  $d$ - or the  $q$ -axis with the  $\alpha$ -axis at time  $t = 0$ .

The figures show the  $\alpha$ - $\beta$  axes components in an  $\alpha\beta$  reference frame and a rotating  $dq$  reference frame for when:

- The  $d$ -axis aligns with the  $\alpha$ -axis.



- The  $q$ -axis aligns with the  $\alpha$ -axis.

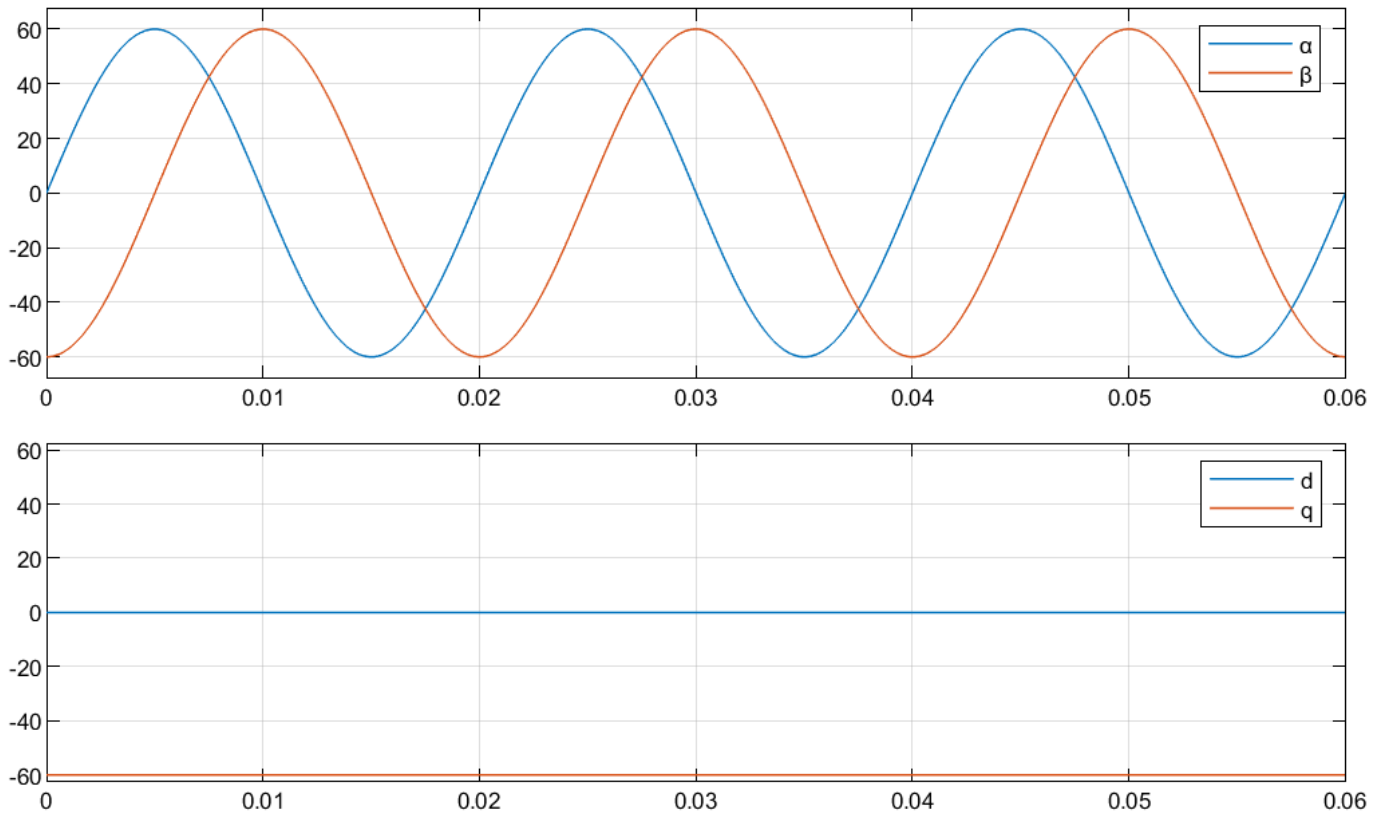


In both cases, the angle  $\theta = \omega t$ , where:

- $\theta$  is the angle between the  $\alpha$ - and  $d$ -axes for the  $d$ -axis alignment or the angle between the  $\alpha$ - and  $q$ -axes for the  $q$ -axis alignment. It indicates the angular position of the rotating  $dq$  reference frame with respect to the  $\alpha$ -axis.
- $\omega$  is the rotational speed of the  $d$ - $q$  reference frame.
- $t$  is the time, in seconds, from the initial alignment.

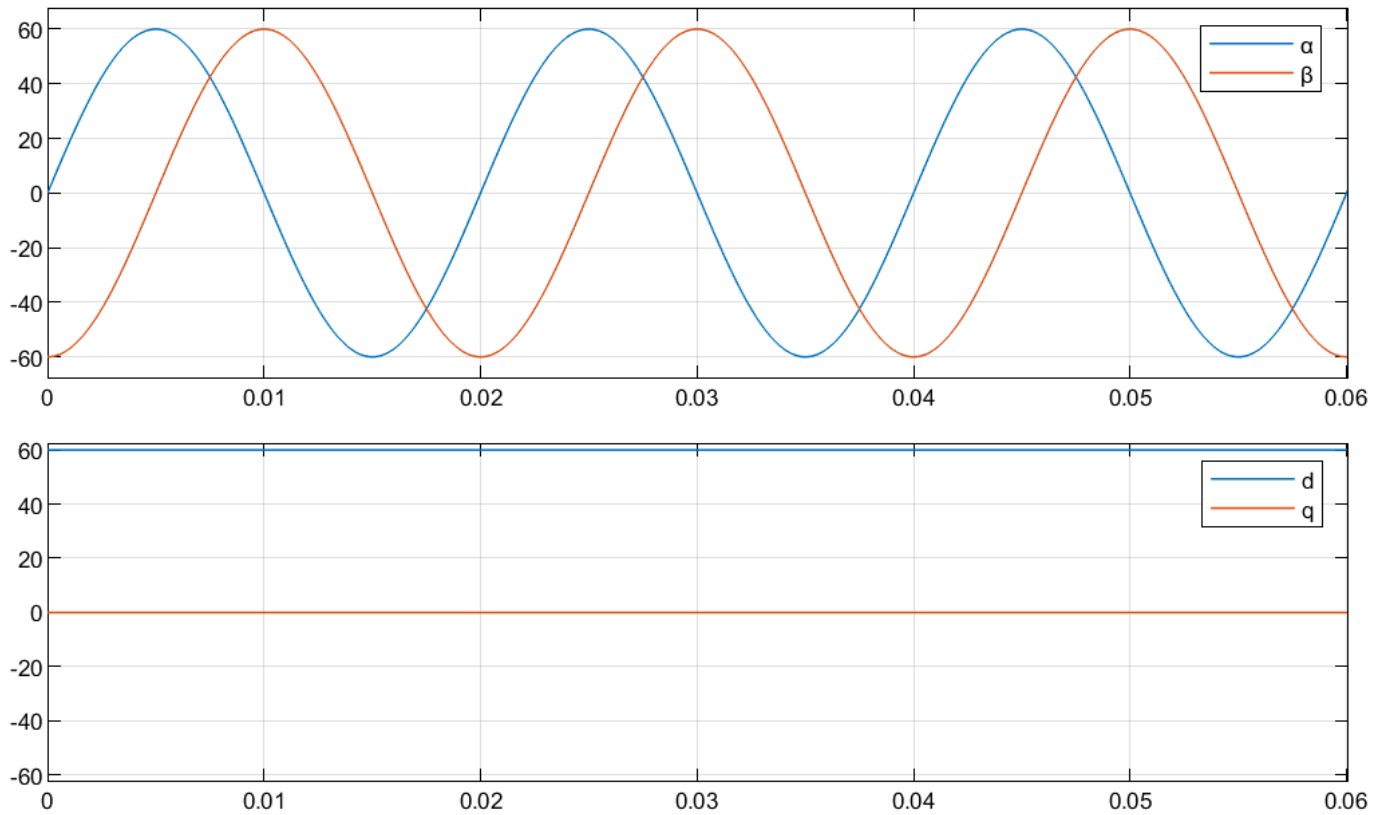
The figures show the time-response of the individual components of the  $\alpha\beta$  and  $dq$  reference frames when:

- The  $d$ -axis aligns with the  $\alpha$ -axis.



- The  $q$ -axis aligns with the  $\alpha$ -axis.





### Equations

The following equations describe how the block implements Park transformation.

- When the  $d$ -axis aligns with the  $\alpha$ -axis.

$$\begin{bmatrix} f_d \\ f_q \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix}$$

- When the  $q$ -axis aligns with the  $\alpha$ -axis.

$$\begin{bmatrix} f_d \\ f_q \end{bmatrix} = \begin{bmatrix} \sin\theta & -\cos\theta \\ \cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} f_\alpha \\ f_\beta \end{bmatrix}$$

where:

- $f_\alpha$  and  $f_\beta$  are the two-phase orthogonal components in the stationary  $\alpha\beta$  reference frame.
- $f_d$  and  $f_q$  are the direct and quadrature axis orthogonal components in the rotating  $dq$  reference frame.

## Ports

### Input

#### **$\alpha$ – Axis component**

scalar

Alpha-axis component,  $\alpha$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$\beta$ – Axis component**

scalar

Beta-axis component,  $\beta$ , in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$\sin \theta_e$ – Sine value of rotational angle**

scalar

Sine value of the angle of transformation,  $\theta_e$ .  $\theta_e$  is the angle between the rotating reference frame and the  $\alpha$ -axis.

Data Types: `single` | `double` | `fixed point`

#### **$\cos \theta_e$ – Cosine value of rotational angle**

scalar

Cosine value of the angle of transformation,  $\theta_e$ .  $\theta_e$  is the angle between the rotating reference frame and the  $\alpha$ -axis.

Data Types: `single` | `double` | `fixed point`

### Output

#### **$d$ – Axis component**

scalar

Direct axis component,  $d$ , in the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **$q$ – Axis component**

scalar

Quadrature axis component,  $q$ , in the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

## Parameters

#### **Alpha (phase-a) axis alignment – $dq$ reference frame alignment**

D-axis (default) | Q-axis

Align either the  $d$ - or  $q$ -axis of the rotating reference frame to the  $\alpha$ -axis of the stationary reference frame.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

HDL Coder provides additional configuration options that affect HDL implementation and synthesized logic.

### HDL Architecture

This block has a single, default HDL architecture.

### HDL Block Properties

<b>ConstrainedOutputPipeline</b>	Number of registers to place at the outputs by moving existing delays within your design. Distributed pipelining does not redistribute these registers. The default is 0. For more details, see “ConstrainedOutputPipeline” (HDL Coder).
<b>InputPipeline</b>	Number of input pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “InputPipeline” (HDL Coder).
<b>OutputPipeline</b>	Number of output pipeline stages to insert in the generated code. Distributed pipelining and constrained output pipelining can move these registers. The default is 0. For more details, see “OutputPipeline” (HDL Coder).

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Inverse Park Transform | Clarke Transform | Sine-Cosine Lookup | Discrete PI Controller with anti-windup and reset | ACIM Feed Forward Control | ACIM Torque Estimator | PMSM Feed Forward Control | PMSM Torque Estimator

### Topics

“Open-Loop and Closed-Loop Control”

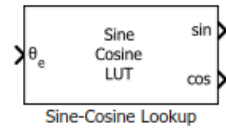
“Field-Oriented Control (FOC)”

### Introduced in R2020a

## Sine-Cosine Lookup

Implement sine and cosine functions using lookup table approach

**Library:** Motor Control Blockset / Controls / Math Transforms

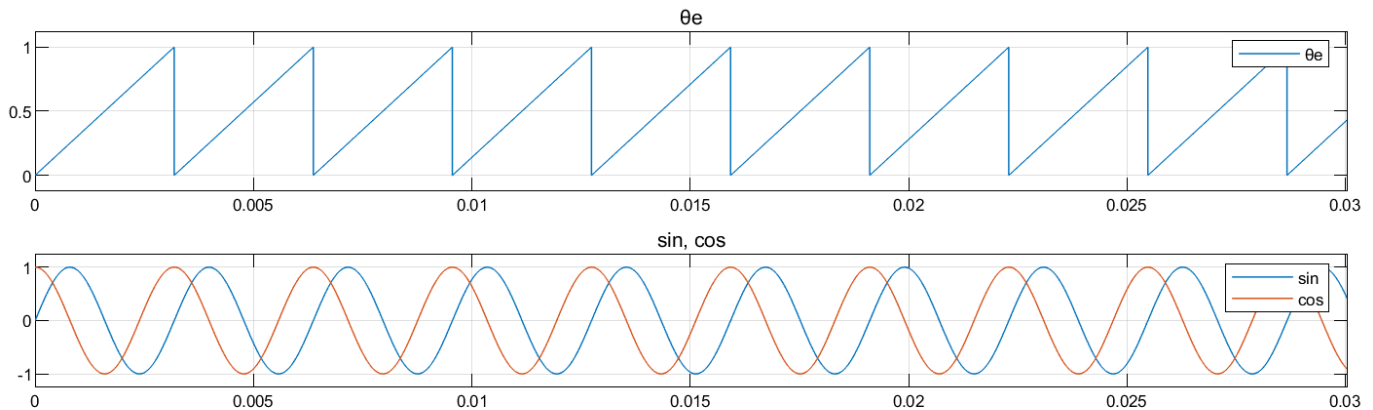


### Description

The Sine-Cosine Lookup block implements sine and cosine functions using the specified position or phase input signal.

The block uses the lookup table approach. This approach results in optimized code-execution when used with the model settings and configuration adopted by the examples shipped in Motor Control Blockset. You can specify the number of lookup table points in the **Number of data points for lookup table** parameter.

This figure shows the input position and the generated sine and cosine output signals:



### Ports

#### Input

##### $\theta_e$ — Reference voltage position

scalar

Position or phase value of the reference voltage signal specified as scalar in either per-unit, radians, or degrees.

Data Types: single | double | fixed point

## Output

### **sin — Sine voltage waveform**

scalar

Sine waveform output with a frequency that is identical to the position or phase signal ( $\theta_e$ ) frequency.

Data Types: `single` | `double` | `fixed point`

### **cos — Cosine voltage waveform**

scalar

Cosine waveform output with a frequency that is identical to the position or phase signal ( $\theta_e$ ) frequency.

Data Types: `single` | `double` | `fixed point`

## Parameters

### **Theta units — Unit of $\theta_e$**

Per-unit (default) | Radians | Degrees

Unit of the input reference voltage position.

### **Number of data points for lookup table — Size of lookup table array**

1024 (default) | scalar

Size of the lookup table array.

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Position Generator | Park Transform | Inverse Park Transform | Mechanical to Electrical Position

### **Topics**

“Open-Loop and Closed-Loop Control”

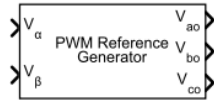
“Field-Oriented Control (FOC)”

### **Introduced in R2020a**

## PWM Reference Generator

Generate modulated signals from phase voltages

**Library:** Motor Control Blockset / Controls / Math Transforms



PWM Reference Generator

### Description

The PWM Reference Generator block generates modulated voltage signals from the stator phase or reference voltages.

The block accepts either the phase voltages ( $V_{abc}$ ) or the stator reference voltages ( $V_{\alpha\beta}$ ) described by the  $\alpha$ - $\beta$  voltage components.

Use this block to perform sinusoidal PWM (SPWM) and space vector modulation (SVM) along with these discrete pulse-width modulation (DPWM) methods that reduce switching losses:

---

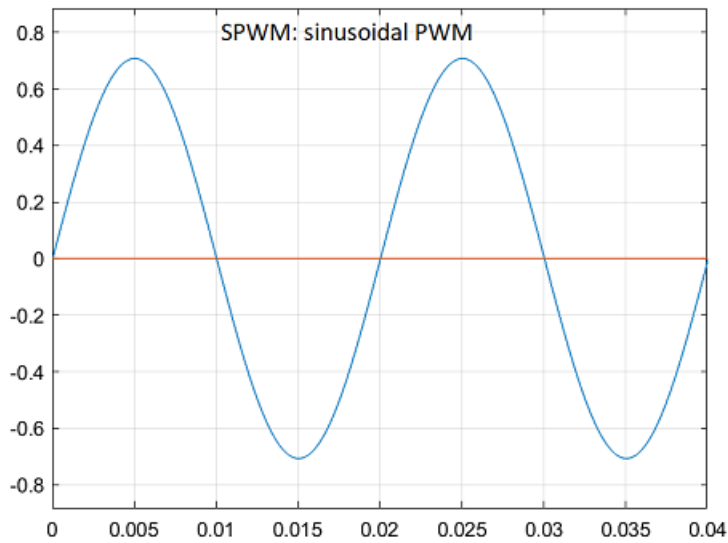
**Note** For the following modulation methods the block supports only per-unit (PU) input signals. For more information about the per-unit system, see “Per-Unit System”.

---

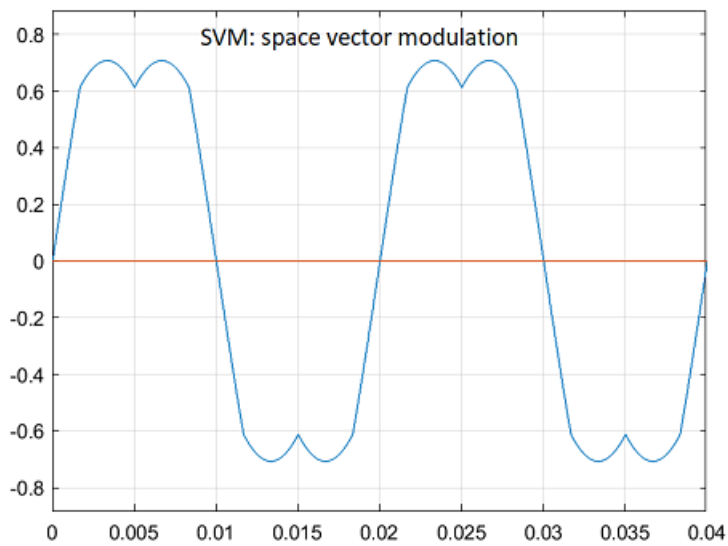
- 60 DPWM — 60 degree discontinuous PWM
- 60 DPWM (+30 degree shift) — +30 degree shift from 60 DPWM
- 60 DPWM (-30 degree shift) — -30 degree shift from 60 DPWM
- 30 DPWM — 30 degree discontinuous PWM
- 120 DPWM — Positive DC component
- 120 DPWM — Negative DC component

For discontinuous PWM (DPWM), the block clamps the modulation wave to the positive or negative DC rail for a total of 120 degrees during each fundamental period per phase. During each clamping interval, the modulation discontinues.

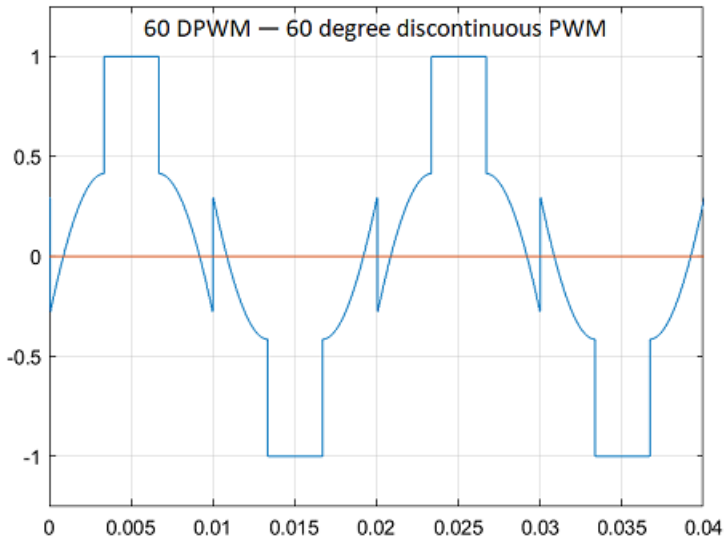
The figure shows the sinusoidal PWM (SPWM) waveform.



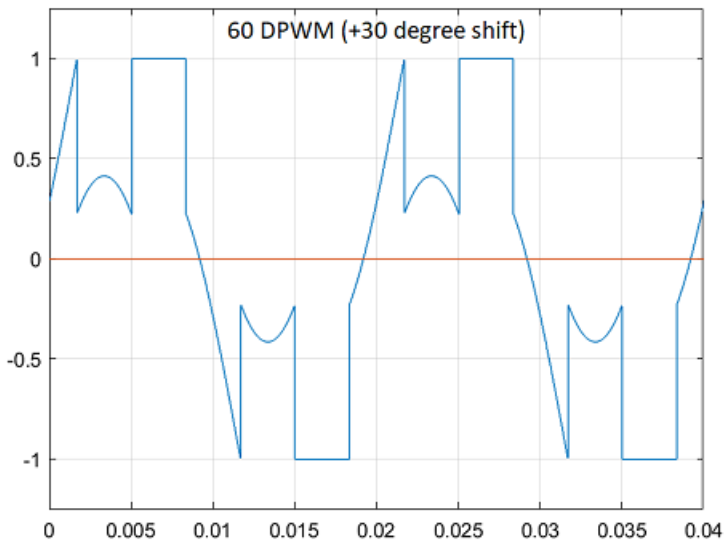
The figure shows the space vector modulation (SVM) waveform.



The figure shows a 60-degree DPWM waveform with two 60-degree clamped intervals per fundamental period.

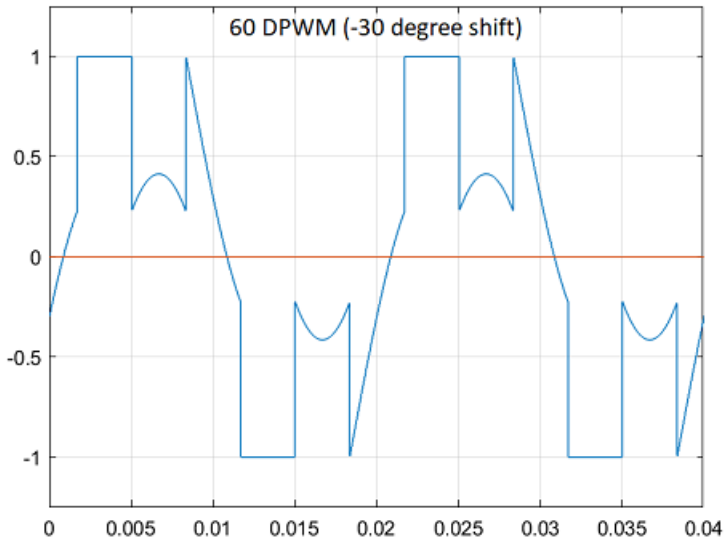


The figure shows a 60-degree DPWM waveform with a positive 30-degree phase shift.

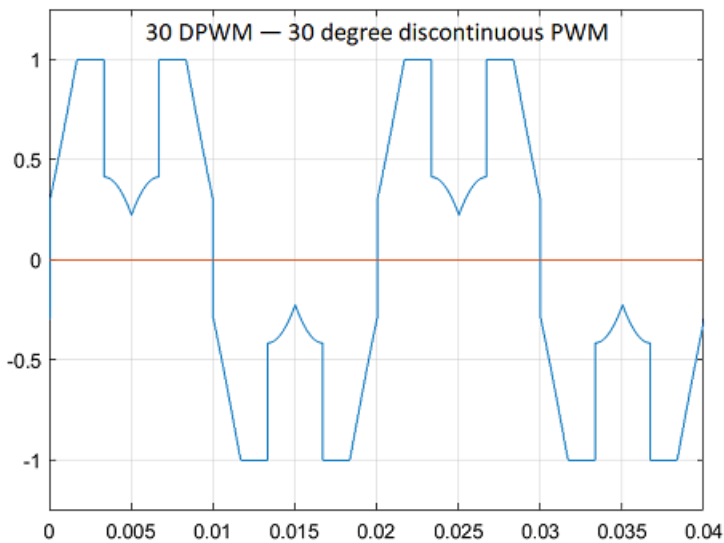


The figure shows a 60-degree DPWM waveform with a negative 30-degree phase shift.

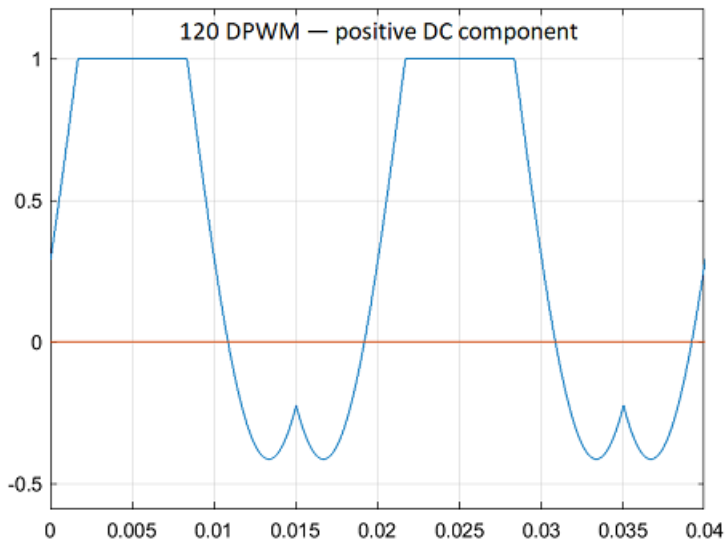




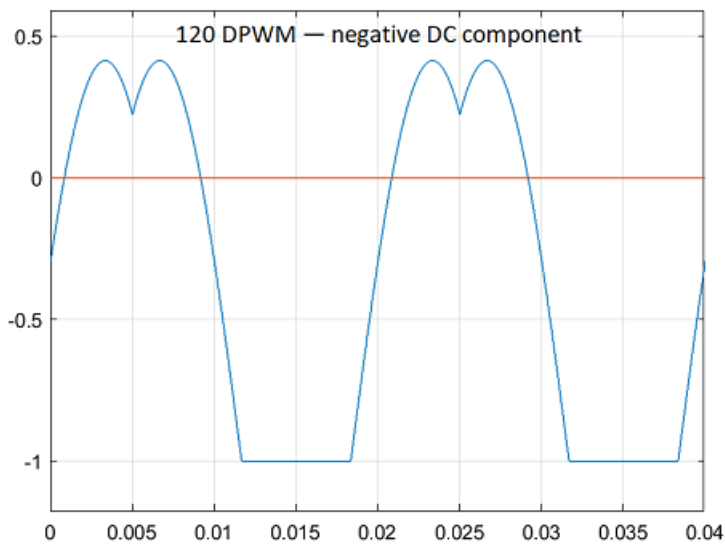
The figure shows a 30-degree DPWM waveform with four 30-degree clamped intervals per fundamental period.



The figure shows a 120-degree DPWM waveform with positive DC clamping.



The figure shows a 120-degree DPWM waveform with negative DC clamping.



## Ports

### Input

#### $V_\alpha$ — $\alpha$ -axis stator reference voltage component

scalar

Stator reference voltage component along  $\alpha$ -axis of the  $\alpha\beta$  reference frame.

#### Dependencies

To enable this port, set **Input type** to Valphabeta.

Data Types: single | double | fixed point

**$V_\beta$  —  $\beta$ -axis stator reference voltage component**

scalar

Stator reference voltage component along  $\beta$ -axis of the  $\alpha\beta$  reference frame.

**Dependencies**

To enable this port, set **Input type** to Valphabeta.

Data Types: single | double | fixed point

 **$V_a$  — Phase component**

scalar

Component of the three-phase system in the  $abc$  reference frame.

**Dependencies**

To enable this port, set **Input type** to Vabc.

Data Types: single | double | fixed point

 **$V_b$  — Phase component**

scalar

Component of the three-phase system in the  $abc$  reference frame.

**Dependencies**

To enable this port, set **Input type** to Vabc.

Data Types: single | double | fixed point

 **$V_c$  — Phase component**

scalar

Component of the three-phase system in the  $abc$  reference frame.

**Dependencies**

To enable this port, set **Input type** to Vabc.

Data Types: single | double | fixed point

**Output** **$V_{a0}$  —  $a$ -axis stator reference voltage component**

scalar

Stator reference voltage component along  $a$ -axis of the  $abc$  reference frame.

Data Types: single | double | fixed point

 **$V_{b0}$  —  $b$ -axis stator reference voltage component**

scalar

Stator reference voltage component along  $b$ -axis of the  $abc$  reference frame.

Data Types: single | double | fixed point

**$V_{co}$  — c-axis stator reference voltage component**

scalar

Stator reference voltage component along *c*-axis of the *abc* reference frame.

Data Types: single | double | fixed point

**Parameters****Input type — Block input type**

Valphabeta (default) | Vabc | Degrees

Type of three-phase stator voltage representation that the block uses as input. Select either the *abc* or *αβ* reference frame.**Modulation method — Pulse-width modulation (PWM) methods**

SVM: space vector modulation (default) | SPWM: sinusoidal PWM | 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

Pulse-width modulation (PWM) method that the block uses to modulate the input stator phase or reference voltages.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

Inverse Park Transform

**Topics**

“Open-Loop and Closed-Loop Control”

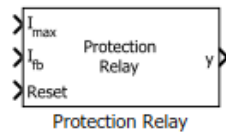
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

# Protection Relay

Implement protection relay with definite minimum time (DMT) trip characteristics

**Library:** Motor Control Blockset / Protection and Diagnostics



## Description

The Protection Relay block implements a protection relay for the hardware and the motor with definite minimum time (DMT) trip characteristics using the reference limit, feedback, and reset input signals. In the event of a fault, the block generates a latched fault signal that you can use to protect the hardware and the motor. You can reset the fault latch using an external reset signal.

## Ports

### Input

**$I_{\max}$  — Upper limit for current**

scalar

Upper limit for current in the feedback loop, so as to provide overcurrent protection. The block generates a latched fault signal when the current in the feedback loop ( $I_{fb}$ ) exceeds this value.

### Dependencies

To enable this port, set **Select Protection** to Overcurrent.

Data Types: single | double | fixed point

**$I_{fb}$  — Actual current in feedback loop**

scalar

Actual current in the feedback loop at a given time.

### Dependencies

To enable this port, set **Select Protection** to Overcurrent.

Data Types: single | double | fixed point

**$\omega_{m \max}$  — Rotor speed limit for overspeed protection**

scalar

Speed limit of the rotor (in RPM). The block generates a latched fault signal when the rotor speed ( $\omega_{fb}$ ) exceeds this value.

### Dependencies

To enable this port, set **Select Protection** to Overspeed.

Data Types: single | double | fixed point

**$\omega_{m\_fb}$  — Actual rotor speed**

scalar

Actual rotor speed at a given time.

**Dependencies**

To enable this port, set **Select Protection** to Overspeed.

Data Types: single | double | fixed point

 **$V_{max}$  — Upper voltage limit for overvoltage protection**

scalar

Upper limit for voltage across the feedback loop. The block generates a latched fault signal when the voltage across the feedback loop ( $V_{fb}$ ) exceeds this value.

**Dependencies**

To enable this port, set **Select Protection** to Overvoltage.

Data Types: single | double | fixed point

 **$V_{min}$  — Lower voltage limit for undervoltage protection**

scalar

Lower limit for voltage across the feedback loop. The block generates a latched fault signal when the voltage across the feedback loop ( $V_{fb}$ ) is less than this value.

**Dependencies**

To enable this port, set **Select Protection** to Undervoltage.

Data Types: single | double | fixed point

 **$V_{fb}$  — Actual voltage across feedback loop**

scalar

Actual voltage across the feedback loop at a given time.

**Dependencies**

To enable this port, set **Select Protection** to either Overvoltage or Undervoltage.

Data Types: single | double | fixed point

**Reset — External reset pulse**

scalar

External pulse that resets the fault latch.

Data Types: single | double | fixed point

**Output** **$y$  — Latched fault signal**

scalar

Latched fault signal that the block generates during the overcurrent, overspeed, overvoltage, and undervoltage conditions to protect the hardware and the motor.

Data Types: single | double | fixed point

## Parameters

### Select Protection — Type of protection relay

Overcurrent (default) | Overspeed | Overvoltage | Undervoltage

Available protection types to configure block behavior during the overcurrent, overspeed, overvoltage, and undervoltage conditions.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

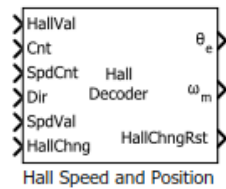
Design and simulate fixed-point systems using Fixed-Point Designer™.

### Introduced in R2020a

## Hall Speed and Position

Compute speed and estimate position of rotor by using Hall sensors

**Library:** Motor Control Blockset / Sensor Decoders



### Description

The Hall Speed and Position block computes the mechanical speed of the rotor by tracking changes in the Hall state. The block also estimates the electric position of the rotor by using the direction, Hall state, and external counter value inputs.

The block executes periodically after a fixed time interval that the controller algorithm defines.

### Ports

#### Input

##### HallVal — Current Hall sensor output state

scalar

The Hall state at current time. For example, these are the possible valid Hall states (where the MSB represents the output of the first Hall sensor connected):

- 5 - (101)
- 4 - (100)
- 6 - (110)
- 2 - (010)
- 3 - (011)
- 1 - (001)

Data Types: single | double | fixed point

##### Cnt — External counter value

scalar

The external counter value that the block uses to determine the time elapsed between the Hall state change and block execution.

Data Types: single | double | fixed point

##### SpdCnt — Count at Hall state change

scalar

This value indicates the clock cycles (time) elapsed between two consecutive changes in the Hall state.



Data Types: single | double | fixed point

### **Dir — Rotor spin direction during current Hall state**

scalar

The direction of the rotor spin (either +1 or -1 indicating positive or negative direction of rotation, respectively) during the current Hall state.

Data Types: single | double | fixed point

### **SpdVal — Validity of current and previous Hall states and speed calculation**

scalar

The port value indicates Hall state validity. The value zero indicates that the current or previous Hall state is invalid and that the block cannot calculate speed and position.

The value one indicates that both the current and previous Hall states are valid and that the block can calculate speed and position.

Data Types: single | double | fixed point

### **HallChng — Value of Hall state change flag**

scalar

The port value indicates Hall state change and block execution status. The value one indicates that the Hall state changed, but that the block execution is pending. The value zero indicates that the block has completed executing the last Hall state change.

Data Types: single | double | fixed point

## **Output**

### **$\theta_e$ — Electrical position of rotor**

scalar

The estimated electrical position of the rotor based on the **Expected hall sequence in positive direction** parameter and the *Direction*, *HallVal*, and *CounterValue* inputs.

#### **Dependencies**

To enable this port, set **Block output** to either *Position* or *Speed and position*.

Data Types: single | double | fixed point

### **$\omega_m$ — Mechanical speed of rotor**

scalar

The mechanical speed of the rotor in revolutions per minute. The block calculates this value by tracking the Hall state changes.

The port returns zero if the *SpdVal* input is zero.

#### **Dependencies**

To enable this port, set **Block output** to either *Speed* or *Speed and position*.

Data Types: single | double | fixed point

**HallChngRst — Resets Hall state change flag to zero**

scalar

The port outputs a value of zero (sets the Hall state change flag to zero) indicating that the block has successfully executed speed and position computations for the last Hall state change.

Data Types: single | double | fixed point

**Parameters****General****Block output — Select output ports**

Speed and position (default) | Speed | Position

Select the available block output ports as one of the following values:

- Speed and position
- Speed
- Position

**Counter size — Size of external counter register**

32 bits (default) | 8 bits | 16 bits

The register size of the external counter. The maximum counter value is  $2^n - 1$ , where  $n$  = counter size.

**Counter clock frequency (Hz) — Clock frequency of external counter**

90e6 (default) | scalar

The clock frequency of the external counter.

**Discrete step size (s) — Sample time after which block executes again**

50e-6 (default) | scalar

The time between two consecutive instances of block execution.

**Number of pole pairs — Number of pole pairs available in motor**

8 (default) | scalar

Number of pole pairs available in the motor.

**Minimum detectable speed (RPM) — Minimum speed that block can detect**

20 (default) | scalar

The block does not calculate position for speed below this value.

**Speed measurement interval — Interval over which block measures speed**

Every 180 Degrees (default) | Every 60 Degrees

Rotor angular displacement that represents the interval at which the SpdCnt port value was calculated.

**Speed unit — Unit of rotor angular speed output**

Radians/sec (default) | Degrees/sec | RPM | Per unit

Unit of the angular velocity or mechanical speed ( $\omega_m$ ) output.

#### Dependencies

To enable this parameter, set **Block output** to either Speed or Speed and position.

#### Speed datatype — Data type of rotor angular speed output

single (default) | double | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

Data type of the rotor angular speed output.

#### Dependencies

To enable this parameter, set **Block output** to either Speed or Speed and position.

#### Position

#### Expected hall sequence in positive direction — Sequence indicating positive direction

ABC (default) | CBA | Custom sequence

The Hall sensor sequence that represents the positive direction of rotor spin.

#### Dependencies

To enable this parameter, set **Block output** to either Position or Speed and position.

#### Sequence — Custom sequence indicating positive direction

[5,4,6,2,3,1] (default) | scalar

The custom sequence that you can enter to represent rotor spin in the positive direction.

#### Dependencies

To enable this parameter:

- Set **Block output** to either Position or Speed and position.
- Set **Expected hall sequence in positive direction** to Custom sequence.

#### Order of extrapolation of position — Indicates precision in position computation

1st Order (default) | 2nd Order

The 1st Order option is less accurate in computing position, but quick. The 2nd Order option is more accurate, but needs more computation time. These equations describe the options:

$$\theta_{1st\ Order} = \theta_{sector} + \omega t$$

$$\theta_{2nd\ Order} = \theta_{sector} + \omega t + \frac{1}{2}\alpha t^2$$

where:

$\theta_{1st\ Order}$  = Position computed by using 1st order extrapolation.

$\theta_{2nd\ Order}$  = Position computed by using 2nd order extrapolation.

$\theta_{sector}$  = Sector angle defined by the Hall sensor output.

$\omega$  = Angular velocity of the rotor.

$\alpha$  = Angular acceleration of the rotor.

$t$  = Time spent in a sector.

#### **Dependencies**

To enable this parameter, set **Block output** to either Position or Speed and position.

#### **Position unit – Unit of angular speed output**

Radians (default) | Degrees | Per unit

Unit of angular speed output.

#### **Dependencies**

To enable this parameter, set **Block output** to either Position or Speed and position.

#### **Position datatype – Data type of angular speed output**

single (default) | double | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

Data type of angular speed output.

#### **Dependencies**

To enable this parameter, set **Block output** to either Position or Speed and position.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

Hall Validity | Mechanical to Electrical Position | Discrete PI Controller with anti-windup and reset

### **Topics**

“Current Sensor ADC Offset and Position Sensor Calibration”

“Open-Loop and Closed-Loop Control”

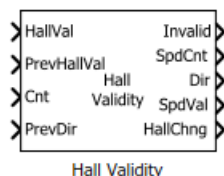
“Field-Oriented Control (FOC)”

### **Introduced in R2020a**

# Hall Validity

Compute rotor spin direction and validity of Hall sensor sequence

**Library:** Motor Control Blockset / Sensor Decoders



## Description

The Hall Validity block checks and validates every state of the Hall sensor output sequence. The block identifies the condition when one or more Hall sensors are in an invalid state.

The block executes when a Hall sensor output state (or Hall state) changes.

## Ports

### Input

#### HallVal — Current Hall sensor output state

scalar

The Hall state at current time. These are the possible input values (three-bit numbers where the MSB represents the output of the first Hall connected):

- 5 - (101)
- 4 - (100)
- 6 - (110)
- 2 - (010)
- 3 - (011)
- 1 - (001)

---

**Note** The output port `Invalid` indicates a bad hall sensor condition.

---

Data Types: `single` | `double` | `fixed point`

#### PrevHallVal — Previous Hall sensor output state

scalar

The Hall state prior to the current state.

Data Types: `single` | `double` | `fixed point`

#### Cnt — External counter value

scalar

The external counter value that the block uses to determine the time elapsed between the Hall state change and block execution.

---

**Note** The counter must reset when a Hall state changes.

---

Data Types: `single` | `double` | `fixed point`

### **PrevDir — Rotor spin direction during previous Hall state**

scalar

The direction of rotor spin (either +1 or -1 indicating positive or negative direction of rotation, respectively) during the previous Hall state.

Data Types: `single` | `double` | `fixed point`

### **Output**

#### **Invalid — Indicator of Hall state validity**

scalar

The indicator of Hall sensor validity during the current or previous Hall state. The block checks the validity of the sensors by comparing the values of the `HallVal` and `PrevHallVal` input port with the value of the **Expected hall sequence in positive direction** parameter. The port can output these values:

- 1 - (001) Indicates that one (or more) sensors are bad.
- 0 - (000) Indicates that all sensors are good.

Data Types: `single` | `double` | `fixed point`

#### **SpdCnt — Count at Hall state change**

scalar

The value of the `Cnt` input port when a Hall state changes.

---

**Note** The counter must reset when a Hall state changes. Therefore, this port indicates the number of counts during the previous Hall state.

---

Data Types: `single` | `double` | `fixed point`

#### **Dir — Rotor spin direction during current Hall state**

scalar

The direction of the rotor spin (either +1 or -1 indicating positive or negative direction of rotation, respectively) during the current Hall state. The block computes the direction by comparing the values of the `HallVal` and `PrevHallVal` input ports with the value of the **Expected hall sequence in positive direction** parameter.

Data Types: `single` | `double` | `fixed point`

#### **SpdVal — Validity of current and previous Hall states and speed calculation**

scalar

The port outputs zero when either one or both conditions occur:

- The block detects a bad hall sensor state (in either HallVal or PrevHallVal input port values).
- The block detects a change in the rotor spin direction.

The zero value indicates that you cannot calculate the valid speed for the current Hall state because the current value of SpdCnt is invalid. The port outputs the value one to indicate that a valid speed calculation is possible.

Data Types: single | double | fixed point

### **HallChng — Set flag to one, indicating Hall state change**

scalar

The port outputs the value one (and sets the Hall state change flag to one) after the Hall state changes and the block has completed execution.

Data Types: single | double | fixed point

## **Parameters**

### **Expected hall sequence in positive direction — Sequence indicating positive direction**

ABC (default) | CBA | Custom sequence

The Hall sensor sequence that represents the positive direction of rotor spin.

### **Sequence — Custom sequence indicating positive direction**

[5, 4, 6, 2, 3, 1] (default) | scalar

The custom sequence that you can enter to represent rotor spin in the positive direction.

### **Dependencies**

To enable this parameter, set **Expected hall sequence in positive direction** to Custom sequence.

### **Counter size — Size of external counter register**

16 bits (default) | 8 bits | 32 bits

The register size of the external counter. The maximum counter value is  $2^n - 1$ , where  $n$  = counter size.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

Hall Speed and Position

### **Topics**

“Current Sensor ADC Offset and Position Sensor Calibration”

“Open-Loop and Closed-Loop Control”  
“Field-Oriented Control (FOC)”

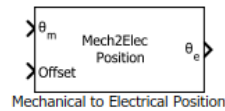
**Introduced in R2020a**



# Mechanical to Electrical Position

Compute electrical position of rotor from mechanical position

**Library:** Motor Control Blockset / Sensor Decoders



## Description

The Mechanical to Electrical Position block computes the electrical position of rotor by using its mechanical position and mechanical offset value.

## Ports

### Input

#### $\theta_m$ — Mechanical position of rotor

scalar

The mechanical position of rotor (as output by the rotor position sensor) in either radians (0 to  $2\pi$ ), degrees (0 to 360), or per unit (0 to 1).

Data Types: `single` | `double` | `fixed point`

#### Offset — Mechanical position offset

scalar

The deviation of the rotor's electrical zero from the mechanical zero position. Unit of offset is identical to the unit of the mechanical position input.

### Dependencies

- To enable this port, set **Specify offset via** to `Input` port.
- Inputs must be of the same data type.

Data Types: `single` | `double` | `fixed point`

### Output

#### $\theta_e$ — Electrical position of rotor

scalar

The electrical position of the rotor with a range that is identical to that of the mechanical position input. Data type of the electrical position is identical to that of the input.

Data Types: `single` | `double` | `fixed point`

## Parameters

#### Number of pole pairs — Number of pole pairs available in motor

4 (default) | scalar

Number of pole pairs available in the motor.

**Input mechanical angle unit – Unit of mechanical position of rotor**

Per unit (default) | Radians | Degrees

Unit of the mechanical position of the rotor.

**Offset input type – Method to specify offset**

Input port (default) | Specify via dialog

The method you want to use to specify the mechanical position offset. Select `Input port` to enable and use the input port `Offset`. Select `Specify via dialog` to provide the offset value using the dialog box.

**Mechanical offset – Value of mechanical position offset**

0 (default) | scalar

The unit of the offset is identical to that of the unit of the mechanical position input.

**Dependencies**

To enable this parameter, set **Specify offset via** to `Specify via dialog`.

**Input data type – Data type of input ports**

single (default) | double | fixed point

The data type that you want to use for the input ports.

---

**Note** The block runs faster, if you select either `fixdt(1,16,0)` or `fixdt(1,16,2^0,0)` input data type and provide fixed point values to the input ports.

---

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Hall Speed and Position | Quadrature Decoder | Position Generator | Discrete PI Controller with anti-windup and reset | Sine-Cosine Lookup

**Topics**

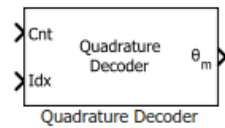
“Open-Loop and Closed-Loop Control”  
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

# Quadrature Decoder

Compute position of quadrature encoder

**Library:** Motor Control Blockset / Sensor Decoders



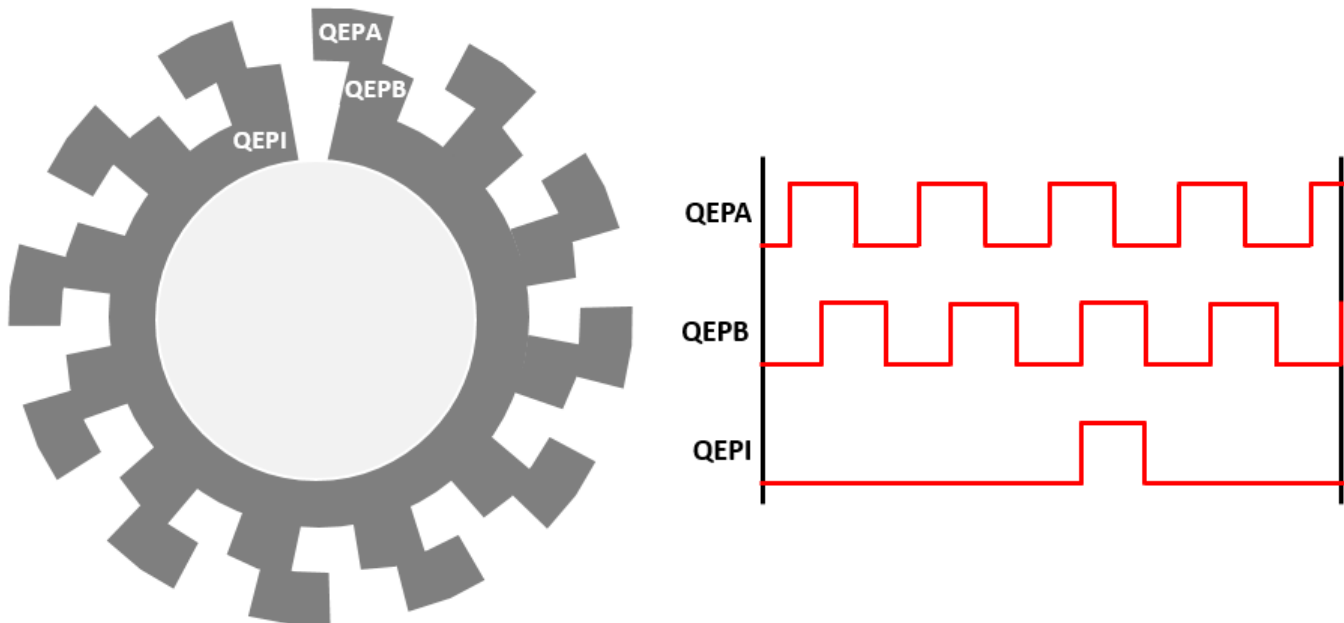
## Description

The Quadrature Decoder block computes the position of the quadrature encoder.

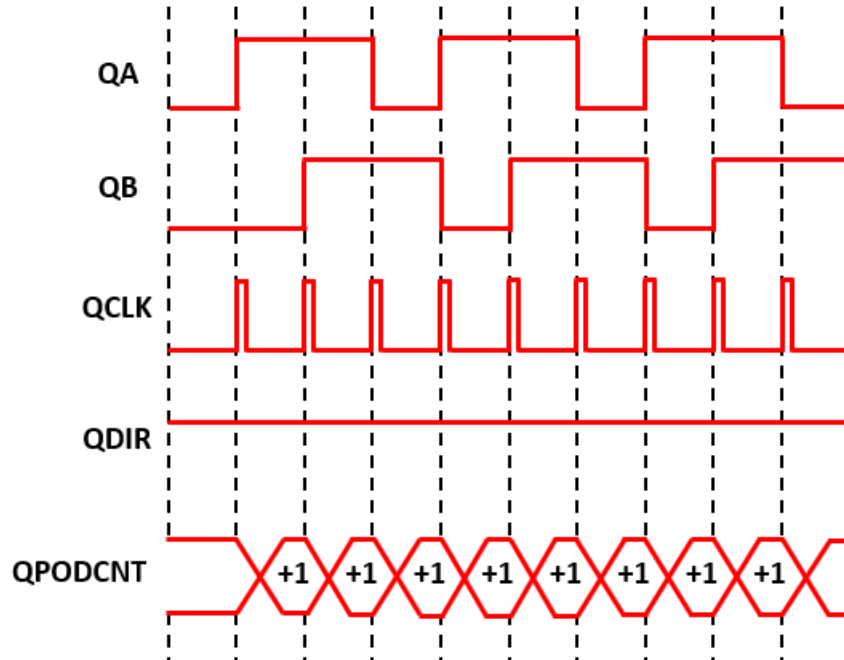
To calculate the angular position of the quadrature encoder (and the rotor) in either degrees, radians, or per-unit, the block uses one of the following methods.

- Difference between current encoder counter value and encoder counter value at the previous index pulse (when index pulse is available).
- Current encoder counter value (when index pulse is not available).

This figure shows a quadrature encoder disk with two channels (QEPA and QEPB) and an index pulse (QEPI):



In this example, the timer driven by the QEP increments by four for each slit:



### Equations

The block computes the angular position (in counts) of the quadrature encoder as:

When the encoder rotates in the clockwise direction:

- If  $Idx \leq Cnt$ ,  

$$Position\ count = (Cnt - Idx)$$
- If  $Idx > Cnt$  and the shaft continues to rotate in the clockwise direction,  

$$Position\ count = (Cnt - Idx)$$
- If  $Idx > Cnt$  and the shaft starts rotating in the anticlockwise direction,  

$$Position\ count = Counts\ per\ revolution - (Idx - Cnt)$$

When the encoder rotates in the anticlockwise direction:

- If  $Idx \geq Cnt$ ,  

$$Position\ count = Counts\ per\ revolution - (Idx - Cnt)$$
- If  $Idx < Cnt$  and the shaft continues to rotate in the anticlockwise direction,  

$$Position\ count = Counts\ per\ revolution - (Idx - Cnt)$$
- If  $Idx < Cnt$  and the shaft starts rotating in the clockwise direction,  

$$Position\ count = (Cnt - Idx)$$

When you clear the **External index count** parameter, the **Idx** pulse resets **Cnt** to zero, therefore:

$$Position\ count = Cnt$$

where:

- *Position count* is the angular position of the quadrature encoder in counts.
- *Counts per revolution* is the number of counts in one rotation cycle of the quadrature encoder.

The block computes the output  $\theta_m$  as:

$Position = 360 \times Position\ count / (Encoder\ slits \times Encoder\ counts\ per\ slit)$  (in degrees)

$Position = 2\pi \times Position\ count / (Encoder\ slits \times Encoder\ counts\ per\ slit)$  (in radians)

$Position = Cnt / (Encoder\ slits \times Encoder\ counts\ per\ slit)$  (in per-unit)

## Ports

### Input

#### **Cnt — Quadrature encoder counter value**

scalar

Value that the quadrature encoder counter generates with respect to the slit-position. The port only accepts a scalar unsigned integer based on the **Counter size** parameter. For example, if you select 8 bits for **Counter size**, the input data type must be uint8.

Data Types: uint8 | uint16 | uint32

#### **Idx — Quadrature encoder counter value at last index pulse**

scalar

Value that the quadrature encoder counter generated with respect to the slit-position at the time of the last index pulse. The port only accepts a scalar unsigned integer based on the **Counter size** parameter. For example, if you select 8 bits for **Counter size**, the input data type must be uint8.

### Dependencies

To enable this port, select the **External index count** parameter.

Data Types: uint8 | uint16 | uint32

---

**Note** The input data types for both Cnt and Idx must be identical.

---

### Output

#### **$\theta_m$ — Angular position of quadrature encoder**

scalar

Angular position that the block computes based on the Cnt and Idx inputs.

Data Types: single | double | fixed point

## Parameters

#### **Encoder slits — Number of slits per phase**

1000 (default) | scalar

The number of slits available in each phase of the quadrature encoder.

**Encoder counts per slit — Number of counts generated for every slit**

4 (default) | 1 | 2

The number of counts that the quadrature encoder generates for every slit. A count indicates a slit position. For example, select 4 (quadrature mode) if you want the encoder to generate four counts corresponding to 00, 10, 11, and 01 slit positions or values. If you select the quadrature mode:

Encoder density, in counts per revolution (post-quadrature) = **Encoder counts per slit (4) × Encoder slits**

**Counter size — Size of quadrature encoder counter**

16 bits (default) | 8 bits | 32 bits

Counter size signifies the register size of the counter used by the processor to count the quadrature encoder pulses. It is the data type of the counter output of eQEP.

**External index count — Enable Idx input port**

on (default) | off

The block enables the Idx input port only if you select this parameter. The block expects that the Cnt input port value resets at the time of the Idx pulse.

**Position unit — Unit of angular position output**

Radians (default) | Degrees | Per unit

Unit of the angular position output.

**Position data type — Data type of angular position output**

single (default) | double | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

The data type for the angular position output.

---

**Note** The Quadrature Decoder block may occasionally display the warning message 'Wrap on overflow detected.'

---

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Speed Measurement | Mechanical to Electrical Position

**Topics**

“Current Sensor ADC Offset and Position Sensor Calibration”

“Open-Loop and Closed-Loop Control”

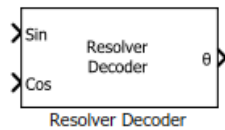
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

## Resolver Decoder

Compute electrical angular position of resolver

**Library:** Motor Control Blockset / Sensor Decoders



### Description

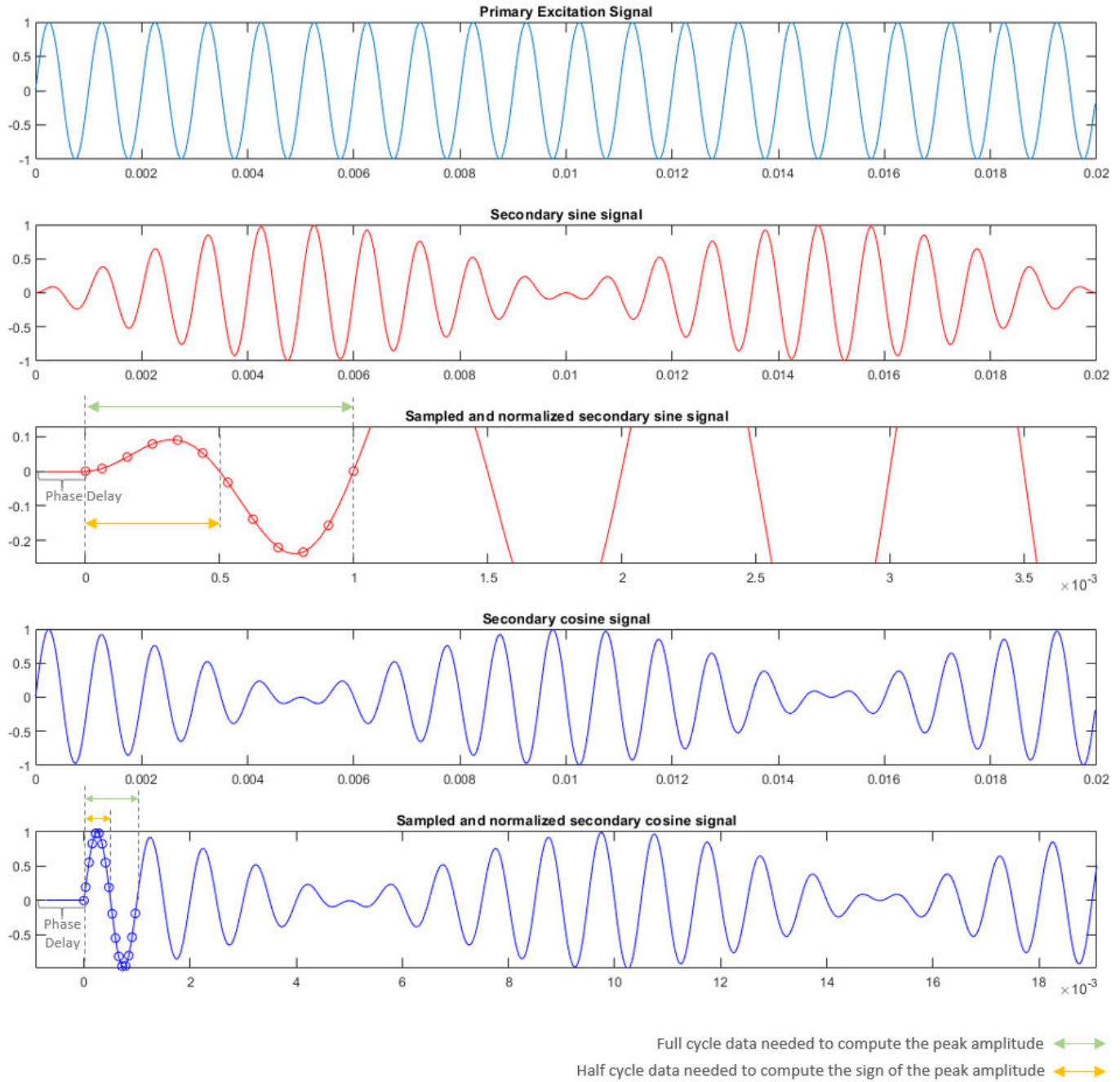
The Resolver Decoder block calculates the electrical angular position of the resolver from the resolver sine and cosine output signals.

The resolver uses a primary sinusoidal excitation input signal to generate the modulated secondary sine and cosine waveforms.

You must normalize these waveforms (within the range of  $[-1,1]$  and centered at 0) and sample them to obtain the secondary sine and cosine input signals of the Resolver Decoder block.

The block computes and outputs the resolver position in  $[0, 2\pi]$  radians. The block can also add a phase delay to the sampled sine and cosine signals with respect to the excitation signal.





**Note** The block inputs should have identical amplitude and data types (either signed fixed or floating point).

## Equations

The block computes the average, peak amplitude values, and the sign of the peak amplitude of a signal cycle as

$$\hat{A}_{average} = \frac{1}{n} \sum_{i=0}^{n-1} (|\hat{A}_i|)$$

$$\hat{A}_{peak} = \hat{A}_{average} \times \frac{\pi}{2}$$

$$\text{Sign of Peak} = \text{Sign of} \left[ \sum_{i = \text{phase delay}}^{\frac{n}{2} - 1 + \text{phase delay}} \hat{A}_i \right]$$

where:

- $\hat{A}_{average}$  is the average amplitude value of a signal cycle
- $n$  is the number of samples per excitation cycle
- $\hat{A}_{peak}$  is the peak amplitude value of a signal cycle

The block computes the electrical angular position of the resolver as

$$\theta = \text{atan2} \frac{u_{\sin\_peak}}{u_{\cos\_peak}}$$

where:

- $u_{\sin\_peak}$  is the  $\hat{A}_{peak}$  of the secondary sine signal
- $u_{\cos\_peak}$  is the  $\hat{A}_{peak}$  of the secondary cosine signal
- $\theta$  is the electrical angular position of the resolver

## Ports

### Input

#### **Sin** — Sampled and normalized secondary sine signal

vector

Secondary sine waveform output from the resolver that is sampled and normalized within the range of [-1, 1] and centered at 0.

Data Types: `single` | `double` | `fixed point`

#### **Cos** — Sampled and normalized secondary cosine signal

vector

Secondary cosine waveform output from the resolver that is sampled and normalized within the range of [-1, 1] and centered at 0.

Data Types: `single` | `double` | `fixed point`

### Output

#### **θ** — Resolver position

scalar

Electrical angular position of the resolver (and the rotor) in  $[0, 2\pi]$  radians.

Data Types: `single` | `double` | `fixed point`

## Parameters

### Phase delay (electrical radians) — Phase delay for input signals

0.1746 (default) | scalar

The phase delay that the block must add to the Sin and Cos input port signals.

### Number of samples per excitation signal — Samples per cycle of input signal

16 (default) | even scalar greater than zero

Number of samples available in one cycle of the Sin and Cos input port signals.

### Output data type — Data type of resolver position output

`single` (default) | `double` | `fixed point`

The data type of the resolver position output  $\theta$ .

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

### Topics

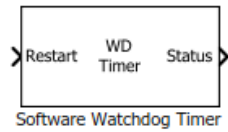
“Monitor Resolver Using Serial Communication”

### Introduced in R2020a

# Software Watchdog Timer

Output true until counter reaches maximum count limit

**Library:** Motor Control Blockset / Sensor Decoders



## Description

The Software Watchdog Timer block increments the counter value until either the block receives a **Restart** input pulse, or the count reaches the value of the **Maximum count** parameter.

On receiving the **Restart** pulse, the block restarts the counter and starts incrementing the counter value again when the **Restart** pulse falls.

The block maintains the true **Status** output until the counter value remains less than the value of **Maximum count** parameter. When the counter reaches **Maximum count**, the block stops the counter and turns the **Status** false.

## Ports

### Input

**Restart — Pulse to restart watchdog timer counter**

scalar

The pulse (true value) that restarts the watchdog timer counter. The counter resumes counting when the pulse falls (false value).

Data Types: `single` | `double` | `fixed point`

### Output

**Status — Watchdog timer status**

scalar

The watchdog timer status indicated as one of the following:

- True indicates that the counter value is less than the value of the **Maximum count** parameter.
- False indicates that the counter value is equal to the value of the **Maximum count** parameter and the block has stopped the counter.

Data Types: `single` | `double` | `fixed point`

## Parameters

**Maximum count — Maximum limit of watchdog timer value**

10 (default) | scalar

The maximum limit of the watchdog timer counter value that causes the block to stop the counter and turn the watchdog timer status to false.

**Counter data type — Data type of Status output**

uint8 (default) | uint16 | uint32

The data type of the watchdog timer status output.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

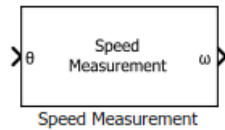
Design and simulate fixed-point systems using Fixed-Point Designer™.

**Introduced in R2020a**

# Speed Measurement

Compute speed from rotor angular position

**Library:** Motor Control Blockset / Sensor Decoders



## Description

The Speed Measurement block calculates the angular speed from the angular position of the rotor by calculating the change in the angular position with respect to time.

## Ports

### Input

#### $\theta$ — Angular position of rotor

scalar

Angular position of the rotor specified in either radians, degrees, or per-unit.

Data Types: `single` | `double` | `fixed point`

### Output

#### $\omega$ — Angular speed of rotor

scalar

Angular speed that the block computes based on the angular position input.

Data Types: `single` | `double` | `fixed point`

## Parameters

#### Position unit — Unit of angular position

Radians (default) | Degrees | Per unit

The unit of the angular position  $\theta$ .

#### Position scaling datatype — Data type of angular position input

`uint32` (default) | `uint16` | `uint64`

The data type of the angular position input  $\theta$ .

#### Speed calculation criteria — Method of speed calculation

Maximum application speed (default) | Speed resolution | Time interval for speed calculation

The speed calculation method used in the block. The selected method determines the range of the rotor speed that the block can measure.

These parameters change values according to the **Speed calculation criteria** parameter:

Parameter name	Maximum application speed	Speed Resolution	Time interval for speed calculation
<b>Delays for speed calculation (number of samples)</b>	299	28	28
<b>Maximum measurable speed (RPM)</b>	1000	10344.8276	10713.2857
<b>Measurable speed resolution (RPM)</b>	4.6566e-07	4.9892e-06	4.9892e-06

#### **Discrete step size (s) – Sample time after which block executes again**

100e-6 (default) | scalar

The fixed time interval (in seconds) between every two consecutive instances of block execution.

These parameters change values according to the **Discrete step size (s)** parameter value:

- **Delays for speed calculation (number of samples)**
- **Maximum measurable speed (RPM)**
- **Measurable speed resolution (RPM)**

#### **Maximum application speed (RPM) – Maximum measurable rotor speed**

1000 (default) | scalar

The maximum rotor speed (in rotations per minute) that the block can measure.

These parameters change values according to the **Maximum application speed (RPM)** parameter value:

- **Delays for speed calculation (number of samples)**
- **Maximum measurable speed (RPM)**
- **Measurable speed resolution (RPM)**

#### **Dependencies**

To enable this parameter, set **Speed calculation criteria** to Maximum application speed.

#### **Speed Resolution (RPM) – Minimum detectable speed**

5e-6 (default) | scalar

The minimum value of change in the  $\theta$  input per unit time that the block can detect.

These parameters change values according to the **Speed Resolution (RPM)** parameter value:

- **Delays for speed calculation (number of samples)**
- **Maximum measurable speed (RPM)**
- **Measurable speed resolution (RPM)**

#### **Dependencies**

To enable this parameter, set **Speed calculation criteria** to Speed resolution.

**Delays for speed calculation (number of samples) – Number of angular position samples measured**

299 or 28 (default) | scalar

The number of samples of the angular position input that the block measures to compute the average position value.

These parameters change values according to the **Delays for speed calculation (number of samples)** parameter value:

- **Maximum measurable speed (RPM)**
- **Measurable speed resolution (RPM)**

**Dependencies**

To enable this parameter, set **Speed calculation criteria** to Time interval for speed calculation.

**Maximum measurable speed (RPM) – Maximum measurable speed**

1000 or 10344.8276 or 10713.2857 (default) | scalar

The absolute maximum speed that the block can measure.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Measurable speed resolution (RPM) – Minimum speed resolution used for speed computation**

4.6566e-07 or 4.9892e-06 (default) | scalar

The minimum speed resolution that the block uses for speed computation. It is always less than or equal to **Speed Resolution (RPM)**.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Speed unit – Unit of angular speed output**

RPM (default) | Degrees/Sec | Radians/Sec | Per unit based on maximum measurable speed | Per unit based on dialog

Unit of the angular speed output.

**Per unit speed (RPM) – Speed (in RPM) for per-unit calculation**

1000 (default)

Specify the speed in RPM for per-unit calculation.

**Dependencies**

This parameter appears only if Per unit based on dialog is selected for **Speed unit**.

**Speed data type – Data type of angular speed output**

single (default) | double | fixed point

The data type of the angular speed output  $\omega$ .



---

**Note** The Speed Measurement block may occasionally display the warning message 'Wrap on overflow detected.'

---

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Quadrature Decoder | Position Generator | Discrete PI Controller with anti-windup and reset | ACIM Feed Forward Control | ACIM Torque Estimator | PMSM Feed Forward Control | PMSM Torque Estimator

### Topics

“Open-Loop and Closed-Loop Control”

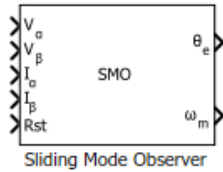
“Field-Oriented Control (FOC)”

### Introduced in R2020a

## Sliding Mode Observer

Compute electrical position and mechanical speed of rotor

**Library:** mcbpositiondecoderlib / Archive



### Description

The Sliding Mode Observer block computes the electrical position and mechanical speed of a PMSM by using the per unit voltage and current values along the  $\alpha$ - and  $\beta$ -axes of the stationary  $\alpha\beta$  reference frame.

### Equations

These equations describe the computation of the electrical position and mechanical speed by the block.

$$\frac{di_{\alpha\beta}}{dt} = \Phi i_{\alpha\beta} + \Gamma V_{\alpha\beta} - \Gamma e_{\alpha\beta}$$

$$i_{\alpha\beta} = [i_{\alpha} \ i_{\beta}]^T$$

$$V_{\alpha\beta} = [V_{\alpha} \ V_{\beta}]^T$$

$$e_{\alpha\beta} = [e_{\alpha} \ e_{\beta}]^T = \begin{bmatrix} -\psi\omega_e \sin\theta_e \\ \psi\omega_e \cos\theta_e \end{bmatrix}$$

$$\Phi = \begin{bmatrix} -\frac{R}{L} & 0 \\ 0 & -\frac{R}{L} \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} \frac{1}{L} & 0 \\ 0 & \frac{1}{L} \end{bmatrix}$$

These equations describe the discrete-time sliding mode observer operation by using per-unit values:

$$\widehat{i}_{\alpha\beta(k+1)P.U} = A \widehat{i}_{\alpha\beta(k)P.U} + \frac{V_{rated}}{I_{rated}} B (v_{\alpha\beta(k)P.U} - \vartheta_{\alpha\beta(k)P.U})$$

$$\vartheta_{\alpha\beta(k+1)P.U} = \vartheta_{\alpha\beta(k)P.U} + 2\pi f_0 \times (\mathbf{Z}(I_{rated}(\widehat{i}_{\alpha\beta(k)P.U} - i_{\alpha\beta(k)P.U})) - \vartheta_{\alpha\beta(k)P.U})$$

$$A = e^{\Phi T_s}$$

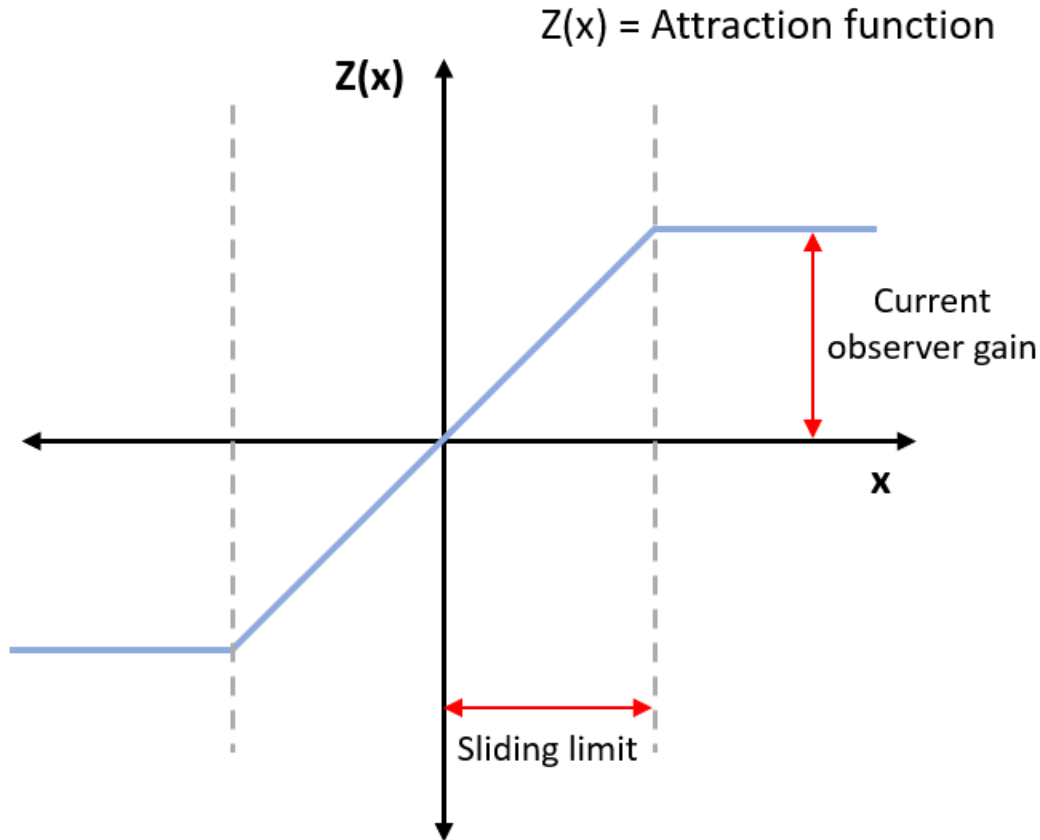
$$B = \int_0^{T_s} e^{\Phi\tau} d\tau$$

$$f_0 = \frac{F_0}{F_s}$$

$$F_s = \frac{1}{T_s}$$

where:

- $e_\alpha, i_\alpha$  are the stator back EMF and current for the  $\alpha$  axis.
- $e_\beta, i_\beta$  are the stator back EMF and current for the  $\beta$  axis.
- $v_\alpha, v_\beta$  are the stator supply voltages.
- $R$  is the stator resistance.
- $L$  is the stator inductance.
- $\psi$  is the flux linkage due to permanent magnet.
- $\omega_e$  is the electrical angular velocity.
- $\theta_e$  is the electrical position of the rotor.
- $t$  is the time.
- $T_s$  is the sampling period.
- $k$  is the sample count.
- $V_{rated}$  is the nominal voltage corresponding to 1 per-unit.
- $I_{rated}$  is the nominal current corresponding to 1 per-unit.
- $Z$  is the attraction function.



- $f_0$  is the cut-off frequency of the filter in cycles per sample.
- $F_0$  is the cut-off frequency in cycles per second.
- $F_s$  is the sample frequency in samples per second.
- $\vartheta_{\alpha\beta(k)}$  is the estimated back EMF.

### Tuning

Use the **Current observer gain** and **Sliding surface limit** parameters to tune the block.

- To improve stability, increase the **Sliding surface limit** or reduce the **Current observer gain**.
- To reduce distortion, decrease the **Current observer gain** or increase the **Sliding surface limit**.

### Ports

#### Input

##### $V_\alpha$ — $\alpha$ -axis voltage

scalar

Per-unit voltage component along the  $\alpha$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: single | double | fixed point

**$V_\beta$  —  $\beta$ -axis voltage**

scalar

Per-unit voltage component along the  $\beta$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: single | double | fixed point

 **$I_\alpha$  —  $\alpha$ -axis current**

scalar

Per-unit current component along the  $\alpha$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: single | double | fixed point

 **$I_\beta$  —  $\beta$ -axis current**

scalar

Per-unit current component along the  $\beta$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: single | double | fixed point

**Rst — Reset the block**

scalar

The pulse (true value) that resets and restarts the processing of the block algorithm.

Data Types: single | double | fixed point

**Output** **$\theta_e$  — Electrical position of PMSM**

scalar

The estimated electrical position of the rotor.

Data Types: single | double | fixed point

 **$\omega_m$  — Mechanical speed of PMSM**

scalar

The estimated mechanical speed of the rotor.

Data Types: single | double | fixed point

**Parameters****Observer parameters****Current observer gain — Sliding mode observer gain for current**

1.1 (default) | scalar

The attraction function gain.

**Sliding surface limit — Maximum limit of sliding surface of SMO**

0.15 (default) | scalar

The boundary layer limit of the attraction function's domain.

**Position unit – Unit of position output**

Radians (default) | Degrees | Per unit

Unit of the position output.

**Position data type – Data type of position output**

single (default) | double | fixdt(1,16) | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

Data type of the position output.

**Speed unit – Unit of speed output**

RPM (default) | Degrees/sec | Radians/sec | Per unit

Unit of the speed output.

**Speed data type – Data type of speed output**

single (default) | double | fixdt(1,16) | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

Data type of the speed output.

**Discrete step size (s) – Sample time after which block executes again**

50e-6 (default) | scalar

The fixed time interval (in seconds) between every two consecutive instances of block execution.

**Motor parameters****Stator resistance (ohm) – Resistance**

0.4836 (default) | scalar

Stator phase winding resistance (in ohm).

**Stator inductance (H) – Inductance**

1e-3 (default) | scalar

Stator phase winding inductance (in Henry).

**Maximum application speed (RPM) – Maximum supported speed value**

6000 (default) | scalar

Maximum value of speed (in RPM) that the block can support. For a speed beyond this value, the block generates incorrect outputs.

**Number of pole pairs – Number of pole pairs available in motor**

4 (default) | scalar

Number of pole pairs available in the motor.

**Base voltage – Nominal voltage corresponding to one per unit**

68 (default) | scalar

The maximum phase voltage applied to PMSM. For details, see “Per-Unit System”.

**Base current — Nominal current corresponding to one per unit**

10 (default) | scalar

The maximum measurable current supplied to PMSM. For details, see “Per-Unit System”.

---

**Note** The Sliding Mode Observer block may occasionally display the warning message 'Wrap on overflow detected.'

---

**References**

- [1] Y. Kung, N. V. Quynh, C. Huang and L. Huang, "Design and simulation of adaptive speed control for SMO-based sensorless PMSM drive," *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012), Kuala Lumpur, 2012*, pp. 439-444 (doi: 10.1109/ICIAS.2012.6306234)
- [2] Zhang Yan and V. Utkin, "Sliding mode observers for electric machines-an overview," *IEEE 2002 28th Annual Conference of the Industrial Electronics Society. IECON 02, Sevilla, 2002*, pp. 1842-1847 vol.3. (doi: 10.1109/IECON.2002.1185251)
- [3] T. Bernardes, V. F. Montagner, H. A. Gründling and H. Pinheiro, "Discrete-Time Sliding Mode Observer for Sensorless Vector Control of Permanent Magnet Synchronous Machine," in *IEEE Transactions on Industrial Electronics*, vol. 61, no. 4, pp. 1679-1691, April 2014 (doi: 10.1109/TIE.2013.2267700)
- [4] Z. Guo and S. K. Panda, "Design of a sliding mode observer for sensorless control of SPMSM operating at medium and high speeds," *2015 IEEE Symposium on Sensorless Control for Electrical Drives (SLED), Sydney, NSW, 2015*, pp. 1-6. (doi: 10.1109/SLED.2015.7339255)

**See Also**

Flux Observer | Clarke Transform | Inverse Park Transform | Sine-Cosine Lookup | Discrete PI Controller with anti-windup and reset

**Topics**

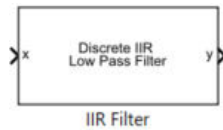
“Open-Loop and Closed-Loop Control”  
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

## IIR Filter

Implement infinite impulse response (IIR) filter

**Library:** Motor Control Blockset / Signal Management



### Description

The IIR Filter block implements a discrete first-order infinite impulse response (IIR) filter on the specified input signal. The block supports fixed-point and floating-point data types. The block is also optimized for code generation when used with the model settings and configuration adopted by the examples shipped in Motor Control Blockset.

We recommend that you use fixed-step discrete solver for this block to enable code generation and ensure accurate simulation.

### Equations

You can configure the IIR filter by using the filter coefficient ( $a$ ) block parameter for a given cutoff frequency ( $f_c$ ).

This equation describes computation of the filter coefficient from the cutoff frequency:

$$a = \left( \frac{2\pi T_s f_c}{2\pi T_s f_c + 1} \right)$$

Alternatively, the block also computes the theoretical cutoff frequency for the given sample time using a filter coefficient:

$$f_c = \left( \frac{a}{(1-a) \cdot 2\pi \cdot T_s} \right)$$

Use the **Filter type** parameter to configure the block either as a low-pass or high-pass filter.

#### Low-pass filter:

$$y(k) = a \cdot x_k + (1 - a) \cdot y_{k-1}$$

#### High-pass filter:

$$y(k) = (1 - a) \cdot x_k - (1 - a) \cdot x_{k-1} + (1 - a) \cdot y_{k-1}$$

where:

- $f_c$  is the cutoff frequency of the IIR filter.
- $a$  is the filter coefficient in the range (0, 1].
- $y(k)$  is the filtered output value at time  $k$ .
- $y_{k-1}$  is the filtered output value at time  $k - 1$ .



- $x_k$  is the sampled input value at time  $k$ .
- $x_{k-1}$  is the filtered output value at time  $k-1$ .
- $T_s$  is the sample time of the IIR Filter block.

## Ports

### Input

#### **x — Sampled input signal**

scalar

Sampled values of the raw input signal in the time domain.

Data Types: single | double | fixed point

### Output

#### **y — Filtered output signal**

scalar

Filtered output signal returned by the IIR Filter block in the time domain.

Data Types: single | double | fixed point

## Parameters

#### **Filter type — IIR filter type**

Low-pass (default) | High-pass

Type of the IIR filter.

#### **Filter co-efficient — Filter coefficient of IIR filter**

0.01 (default) | scalar in the range (0,1]

Filter coefficient of the IIR filter. The data type of this parameter is the same as that of the input signal. We suggest that you check the precision of the parameter value in this data type.

#### **Display cutoff frequency — Display the cutoff frequency parameters**

off (default) | on

Select this parameter for the block to display the **Discrete step size (s)** and **Theoretical cutoff frequency (Hz)** parameters.

#### **Discrete step size (s) — Step size of discrete-time filter**

50e-6 (default) | scalar

Step size of the discrete-time computation (in seconds) used by the IIR filter.

### Dependencies

To display this parameter, select the **Display cutoff frequency** parameter.

#### **Theoretical cutoff frequency (Hz) — Theoretical cutoff frequency of IIR filter**

32.1525 | scalar

Theoretical cutoff frequency (in Hertz) of the IIR filter. This parameter is not configurable.

**Dependencies**

To display this parameter, select the **Display cutoff frequency** parameter.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

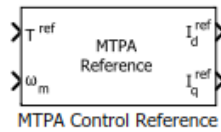
Design and simulate fixed-point systems using Fixed-Point Designer™.

**Introduced in R2020a**

## MTPA Control Reference

Compute reference currents for Maximum Torque Per Ampere (MTPA) and field-weakening operation

**Library:** Motor Control Blockset / Controls / Control Reference



### Description

The MTPA Control Reference block computes the  $d$ -axis and  $q$ -axis reference current values for maximum torque per ampere (MTPA) and field-weakening operations. The computed reference current values results in efficient output for the permanent magnet synchronous motor (PMSM).

The block accepts the reference torque and feedback mechanical speed and outputs the corresponding  $d$ - and  $q$ -axes reference current values for MTPA and field-weakening operations.

The block computes the reference current values by solving mathematical relationships. The calculations use SI unit system. When working with the Per-Unit (PU) system, the block converts PU input signals to SI units to perform computations, and converts them back to PU values at the output.

These equations describe the computation of reference  $d$ -axis and  $q$ -axis current values by the block:

### Mathematical Model of PMSM

These model equations describe dynamics of PMSM in the rotor flux reference frame:

$$v_d = i_d R_s + \frac{d\lambda_d}{dt} - \omega_e L_q i_q$$

$$v_q = i_q R_s + \frac{d\lambda_q}{dt} + \omega_e L_d i_d + \omega_e \lambda_{pm}$$

$$\lambda_d = L_d i_d + \lambda_{pm}$$

$$\lambda_q = L_q i_q$$

$$T_e = \frac{3}{2} p (\lambda_{pm} i_q + (L_d - L_q) i_d i_q)$$

$$T_e - T_L = J \frac{d\omega_m}{dt} + B \omega_m$$

where:

- $v_d$  is the  $d$ -axis voltage (Volts).
- $v_q$  is the  $q$ -axis voltage (Volts).
- $i_d$  is the  $d$ -axis current (Amperes).
- $i_q$  is the  $q$ -axis current (Amperes).
- $R_s$  is the stator phase winding resistance (Ohms).

- $\lambda_{pm}$  is the permanent magnet flux linkage (Weber).
- $\lambda_d$  is the  $d$ -axis flux linkage (Weber).
- $\lambda_q$  is the  $q$ -axis flux linkage (Weber).
- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $\omega_m$  is the rotor mechanical speed (Radians/ sec).
- $L_d$  is the  $d$ -axis winding inductance (Henry).
- $L_q$  is the  $q$ -axis winding inductance (Henry).
- $T_e$  is the electromechanical torque produced by the PMSM (Nm).
- $T_L$  is the load torque (Nm).
- $p$  is the number of motor pole pairs.
- $J$  is the inertia coefficient (kg-m<sup>2</sup>).
- $B$  is the friction coefficient (kg-m<sup>2</sup>/ sec).

### Base Speed

Base speed is the maximum motor speed at the rated voltage and rated load, outside the field-weakening region. These equations describe the computation of the motor base speed.

The inverter voltage constraint is defined by computing the  $d$ -axis and  $q$ -axis voltages:

$$v_{do} = -\omega_e L_q i_q$$

$$v_{qo} = \omega_e (L_d i_d + \lambda_{pm})$$

$$v_{max} = \frac{v_{dc}}{\sqrt{3}} - R_s i_{max} \geq \sqrt{v_{do}^2 + v_{qo}^2}$$

The current limit circle defines the current constraint which can be considered as:

$$i_{max}^2 = i_d^2 + i_q^2$$

In the preceding equation,  $i_d$  is zero for surface PMSMs. For interior PMSMs, values of  $i_d$  and  $i_q$  corresponding to MTPA are considered.

Using the preceding relationships, we can compute the base speed as:

$$\omega_{base} = \frac{1}{p} \cdot \frac{v_{max}}{\sqrt{(L_q i_q)^2 + (L_d i_d + \lambda_{pm})^2}}$$

where:

- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $\omega_{base}$  is the mechanical base speed of the motor (Radians/ sec).
- $i_d$  is the  $d$ -axis current (Amperes).
- $i_q$  is the  $q$ -axis current (Amperes).
- $v_{do}$  is the  $d$ -axis voltage when  $i_d$  is zero (Volts).

- $v_{q0}$  is the  $q$ -axis voltage when  $i_q$  is zero (Volts).
- $L_d$  is the  $d$ -axis winding inductance (Henry).
- $L_q$  is the  $q$ -axis winding inductance (Henry).
- $R_s$  is the stator phase winding resistance (Ohms).
- $\lambda_{pm}$  is the permanent magnet flux linkage (Weber).
- $v_d$  is the  $d$ -axis voltage (Volts).
- $v_q$  is the  $q$ -axis voltage (Volts).
- $v_{max}$  is the maximum fundamental line to neutral voltage (peak) supplied to the motor (Volts).
- $v_{dc}$  is the dc voltage supplied to the inverter (Volts).
- $i_{max}$  is the maximum phase current (peak) of the motor (Amperes).
- $p$  is the number of motor pole pairs.

### Surface PMSM

For a surface PMSM, you can achieve maximum torque by using zero  $d$ -axis current when the motor is below the base speed. For field-weakening operation, the reference  $d$ -axis current is computed by constant-voltage-constant-power control (CVCP) algorithm defined by these equations:

If  $\omega_m \leq \omega_{base}$ :

- $i_{d\_mtpa} = 0$
- $i_{q\_mtpa} = \frac{T^{ref}}{\frac{3}{2} \cdot p \cdot \lambda_{pm}}$
- $i_{d\_sat} = i_{d\_mtpa} = 0$
- $i_{q\_sat} = \text{sat}(i_{q\_mtpa}, i_{max})$

If  $\omega_m > \omega_{base}$ :

- $i_{d\_fw} = \frac{(\omega_{e\_base} - \omega_e)\lambda_{pm}}{\omega_e L_d}$
- $i_{d\_sat} = \max(i_{d\_fw}, -i_{max})$
- $i_{q\_fw} = \frac{T^{ref}}{\frac{3}{2} \cdot p \cdot \lambda_{pm}}$
- $i_{q\_lim} = \sqrt{i_{max}^2 - i_{d\_sat}^2}$
- $i_{q\_sat} = \text{sat}(i_{q\_fw}, i_{q\_lim})$

The saturation function used to compute  $i_{q\_sat}$  is described below:

If  $i_{q\_fw} < -i_{q\_lim}$ ,

$$i_{q\_sat} = -i_{q\_lim}$$

If  $i_{q\_fw} > i_{q\_lim}$ ,

$$i_{q\_sat} = i_{q\_lim}$$

If  $-i_{q\_lim} \leq i_{q\_fw} \leq i_{q\_lim}$ ,

$$i_{q\_sat} = i_{q\_fw}$$

The block outputs the following values:

$$I_d^{ref} = i_{d\_sat}$$

$$I_q^{ref} = i_{q\_sat}$$

where:

- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $\omega_m$  is the rotor mechanical speed (Radians/ sec).
- $\omega_{base}$  is the mechanical base speed of the motor (Radians/ sec).
- $\omega_{e\_base}$  is the electrical base speed of the motor (Radians/ sec).
- $i_{d\_mtpa}$  is the  $d$ -axis phase current corresponding to MTPA (Amperes).
- $i_{q\_mtpa}$  is the  $q$ -axis phase current corresponding to MTPA (Amperes).
- $T^{ref}$  is the reference torque (Nm).
- $p$  is the number of motor pole pairs.
- $\lambda_{pm}$  is the permanent magnet flux linkage (Weber).
- $i_{d\_fw}$  is the  $d$ -axis field weakening current (Amperes).
- $i_{q\_fw}$  is the  $q$ -axis field weakening current (Amperes).
- $L_d$  is the  $d$ -axis winding inductance (Henry).
- $i_{max}$  is the maximum phase current (peak) of the motor (Amperes).
- $i_{d\_sat}$  is the  $d$ -axis saturation current (Amperes).
- $i_{q\_sat}$  is the  $q$ -axis saturation current (Amperes).
- $I_d^{ref}$  is the  $d$ -axis current corresponding to the reference torque and reference speed (Amperes).
- $I_q^{ref}$  is the  $q$ -axis current corresponding to the reference torque and reference speed (Amperes).

### Interior PMSM

For an interior PMSM, you can achieve maximum torque by computing the  $d$ -axis and  $q$ -axis reference currents from the torque equation. For field-weakening operation, the reference  $d$ -axis current is computed by voltage and current limited maximum torque control (VCLMT) algorithm.

The reference currents for MTPA and field weakening operations are defined by these equations:

$$i_{m\_ref} = \frac{2 \cdot T^{ref}}{3 \cdot p \cdot \lambda_{pm}}$$

$$i_m = \min(i_{m\_ref}, i_{max})$$

$$i_{d\_mtpa} = \frac{\lambda_{pm}}{4(L_q - L_d)} - \sqrt{\frac{\lambda_{pm}^2}{16(L_q - L_d)^2} + \frac{i_m^2}{2}}$$

$$i_{q\_mtpa} = \sqrt{i_m^2 - (i_{d\_mtpa})^2}$$

$$v_{d0} = -\omega_e L_q i_q$$

$$v_{q0} = \omega_e (L_d i_d + \lambda_{pm})$$

$$v_{d0}^2 + v_{q0}^2 = v_{max}^2$$

$$(L_q i_q)^2 + (L_d i_d + \lambda_{pm})^2 \leq \frac{v_{max}^2}{\omega_e^2}$$

$$i_q = \sqrt{i_{max}^2 - i_d^2}$$

$$(L_d^2 - L_q^2) i_d^2 + 2\lambda_{pm} L_d i_d + \lambda_{pm}^2 + L_q^2 i_{max}^2 - \frac{v_{max}^2}{\omega_e^2} = 0$$

$$i_{d\_fw} = \frac{-\lambda_{pm} L_d + \sqrt{(\lambda_{pm} L_d)^2 - (L_d^2 - L_q^2) \left( \lambda_{pm}^2 + L_q^2 i_{max}^2 - \frac{v_{max}^2}{\omega_e^2} \right)}}{(L_d^2 - L_q^2)}$$

$$i_{q\_fw} = \sqrt{i_{max}^2 - i_{d\_fw}^2}$$

If  $\omega_m \leq \omega_{base}$ ,

$$I_d^{ref} = i_{d\_mtpa}$$

$$I_q^{ref} = i_{q\_mtpa}$$

If  $\omega_m > \omega_{base}$ ,

$$I_d^{ref} = \max(i_{d\_fw}, -i_{max})$$

$$i_{q\_fw} = \sqrt{i_{max}^2 - i_{d\_fw}^2}$$

If  $i_{q\_fw} < i_m$ ,

$$I_q^{ref} = i_{q\_fw}$$

If  $i_{q\_fw} \geq i_m$ ,

$$I_q^{ref} = i_m$$

For negative reference torque values, the sign of  $i_m$  and  $I_q^{ref}$  are updated and equations are modified accordingly.

where:

- $i_{m\_ref}$  is the estimated maximum current to produce the reference torque (Amperes).
- $i_m$  is the saturated value of estimated maximum current (Amperes).
- $i_{d\_max}$  is the maximum  $d$ -axis phase current (peak) (Amperes).
- $i_{q\_max}$  is the maximum  $q$ -axis phase current (peak) (Amperes).
- $T^{ref}$  is the reference torque (Nm).
- $I_d^{ref}$  is the  $d$ -axis current component corresponding to the reference torque and reference speed (Amperes).
- $I_q^{ref}$  is the  $q$ -axis current component corresponding to the reference torque and reference speed (Amperes).
- $p$  is the number of motor pole pairs.
- $\lambda_{pm}$  is the permanent magnet flux linkage (Weber).
- $i_{d\_mtpa}$  is the  $d$ -axis phase current corresponding to MTPA (Amperes).
- $i_{q\_mtpa}$  is the  $q$ -axis phase current corresponding to MTPA (Amperes).
- $L_d$  is the  $d$ -axis winding inductance (Henry).
- $L_q$  is the  $q$ -axis winding inductance (Henry).
- $i_{max}$  is the maximum phase current (peak) of the motor (Amperes).
- $v_{max}$  is the maximum fundamental line to neutral voltage (peak) supplied to the motor (Volts).
- $v_{d0}$  is the  $d$ -axis voltage when  $i_d$  is zero (Volts).
- $v_{q0}$  is the  $q$ -axis voltage when  $i_q$  is zero (Volts).
- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $i_d$  is the  $d$ -axis current (Amperes).
- $i_q$  is the  $q$ -axis current (Amperes).
- $i_{d\_fw}$  is the  $d$ -axis field weakening current (Amperes).
- $i_{q\_fw}$  is the  $q$ -axis field weakening current (Amperes).
- $\omega_{base}$  is the mechanical base speed of the motor (Radians/ sec).

## Ports

### Input

#### $T^{ref}$ — Reference torque value

scalar

Reference torque input value for which the block computes the reference current.

Data Types: single | double | fixed point

#### $\omega_m$ — Mechanical speed

scalar



Reference mechanical speed value for which the block computes the reference current.

Data Types: `single` | `double` | `fixed point`

## Output

### $I_d^{\text{ref}}$ — Reference *d*-axis current

scalar

Reference *d*-axis phase current that can efficiently generate the input torque and speed values.

Data Types: `single` | `double` | `fixed point`

### $I_q^{\text{ref}}$ — Reference *q*-axis current

scalar

Reference *q*-axis phase current that can efficiently generate the input torque and speed values.

Data Types: `single` | `double` | `fixed point`

## Parameters

### Type of motor — Type of PMSM

Interior PMSM (default) | Surface PMSM

Type of PMSM based on the location of the permanent magnets.

### Number of pole pairs — Number of available pole pairs

4 (default) | scalar

Number of pole pairs available in the motor.

### Stator resistance per phase (Ohm) — Resistance of stator phase winding (ohms)

0.36 (default) | scalar

Resistance of the stator phase winding (ohms).

## Dependencies

To enable this parameter, set **Type of motor** to Interior PMSM.

### Stator *d*-axis inductance (H) — *d*-axis stator winding inductance

0.2e-3 (default) | scalar

Stator winding inductance (henry) along the *d*-axis of the rotating *dq* reference frame.

### Stator *q*-axis inductance (H) — *q*-axis stator winding inductance

0.4e-3 (default) | scalar

Stator winding inductance (henry) along the *q*-axis of the rotating *dq* reference frame.

## Dependencies

To enable this parameter, set **Type of motor** to Interior PMSM.

### Permanent magnet flux linkage (Wb) — Magnetic flux linkage of permanent magnets

6.4e-3 (default) | scalar

Magnetic flux linkage between the stator windings and permanent magnets on the rotor (weber).

**Max current (A) — Maximum phase current limit for motor (amperes)**

7.1 (default) | scalar

Maximum phase current limit for the motor (amperes).

**DC voltage (V) — DC bus voltage (volts)**

24 (default) | scalar

DC bus voltage (volts)

**Dependencies**

To enable this parameter, set **Type of motor** to Interior PMSM.

**Input signal units — Unit of block input values**

Per-Unit (PU) (default) | SI Units

Unit of the block input values.

**Base speed (rpm) — Base speed of motor (rpm)**

4107 (default) | scalar

Speed of the motor at the rated voltage and rated current outside the field weakening region.

**Base current (A) — Base current for per-unit conversion (amperes)**

19.3 (default) | scalar

Current corresponding to 1 per-unit. We recommend that you use the maximum current detected by an Analog to Digital Converter (ADC) as the base current.

**Dependencies**

To enable this parameter, set **Input signal units** to Per-Unit (PU).

**Base torque (Nm) — Base torque for per-unit conversion (Nm)**

0.74112 (default) | scalar

Torque corresponding to 1 per-unit. See “Per-Unit System” page for more details.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Dependencies**

To display this parameter, set **Input signal units** to Per-Unit (PU).

**References**

- [1] *B. Bose, Modern Power Electronics and AC Drives. Prentice Hall, 2001. ISBN-0-13-016743-6.*
- [2] *Morimoto, Shigeo, Masayuka Sanada, and Yoji Takeda. "Wide-speed operation of interior permanent magnet synchronous motors with high-performance current regulator." IEEE Transactions on Industry Applications, Vol. 30, Issue 4, July/August 1994, pp. 920-926.*

- [3] Li, Muyang. "Flux-Weakening Control for Permanent-Magnet Synchronous Motors Based on Z-Source Inverters." *Master's Thesis, Marquette University, e-Publications@Marquette, Fall 2014.*
- [4] Briz, Fernando, Michael W. Degner, and Robert D. Lorenz. "Analysis and design of current regulators using complex vectors." *IEEE Transactions on Industry Applications, Vol. 36, Issue 3, May/June 2000, pp. 817-825.*
- [5] Lorenz, Robert D., Thomas Lipo, and Donald W. Novotny. "Motion control with induction motors." *Proceedings of the IEEE, Vol. 82, Issue 8, August 1994, pp. 1215-1240.*
- [6] Briz, Fernando, et al. "Current and flux regulation in field-weakening operation [of induction motors]." *IEEE Transactions on Industry Applications, Vol. 37, Issue 1, Jan/Feb 2001, pp. 42-50.*
- [7] TI Application Note, "Sensorless-FOC With Flux-Weakening and MTPA for IPMSM Motor Drives."

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Discrete PI Controller with anti-windup and reset

### Topics

"Open-Loop and Closed-Loop Control"

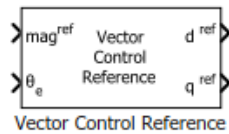
"Field-Oriented Control (FOC)"

### Introduced in R2020a

## Vector Control Reference

Compute  $d$  and  $q$  axis components of reference vector

**Library:** Motor Control Blockset / Controls / Control Reference



### Description

The Vector Control Reference block calculates the  $d$ -axis and  $q$ -axis components of the reference voltage, current, or flux vector that you provide as an input to the block.

The block accepts magnitude and position of the reference vector as inputs. The block outputs the reference vector components along the direct and quadrature axes of the rotating  $dq$  reference frame.

### Equations

The block uses these equations to compute the  $d$ -axis and  $q$ -axis vector component outputs.

$$d^{ref} = mag^{ref} \times \cos\theta_e$$

$$q^{ref} = mag^{ref} \times \sin\theta_e$$

where:

- $d^{ref}$  is the  $d$ -axis component of the reference vector.
- $q^{ref}$  is the  $q$ -axis component of the reference vector.
- $mag^{ref}$  is the magnitude of the reference vector.
- $\theta_e$  is the electrical position of the reference vector.

### Ports

#### Input

**$mag^{ref}$  — Magnitude of reference vector**

scalar

Magnitude of the reference voltage, current, or flux vector that you provide as an input to the block.

Data Types: `single` | `double` | `fixed point`

**$\theta_e$  — Electrical position of reference vector**

scalar

Electrical position of the reference voltage, current, or flux vector that you provide as an input to the block.

Data Types: `single` | `double` | `fixed point`

## Output

### **$d^{\text{ref}}$ — $d$ -axis component of reference vector**

scalar

Reference voltage, current, or flux vector component along the direct axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

### **$q^{\text{ref}}$ — $q$ -axis component of reference vector**

scalar

Reference voltage, current, or flux vector component along the quadrature axis of the rotating  $dq$  reference frame.

Data Types: `single` | `double` | `fixed point`

## Parameters

### **Alpha (phase-a) axis alignment — $dq$ reference frame alignment**

D-axis (default) | Q-axis

Align either the  $d$ - or  $q$ -axis of the rotating reference frame to the  $\alpha$ -axis of the stationary reference frame.

### **Theta units — Unit of input position value**

Per-unit (default) | Radians | Degrees

Unit of the input electrical position of the reference voltage, current, or flux vector.

### **Number of data points for trigonometric lookup table — Size of lookup table array**

1024 (default) | scalar

Size of the lookup table array.

## Extended Capabilities

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Speed Measurement | MTPA Control Reference

### **Topics**

“Open-Loop and Closed-Loop Control”

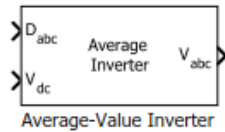
“Field-Oriented Control (FOC)”

**Introduced in R2020a**

# Average-Value Inverter

Compute three-phase AC voltage from inverter DC voltage

**Library:** Motor Control Blockset / Electrical Systems / Inverters



## Description

The Average-Value Inverter block models an average-value and full-wave inverter. It computes the three-phase AC voltage output from inverter DC voltage by using the duty cycle information.

## Equations

These equations describe how the block computes the three-phase AC voltage.

$$D_0 = \frac{(D_a + D_b + D_c)}{3}$$

$$V_a = V_{dc} \times (D_a - D_0)$$

$$V_b = V_{dc} \times (D_b - D_0)$$

$$V_c = V_{dc} \times (D_c - D_0)$$

where:

- $D_a$ ,  $D_b$ , and  $D_c$  are the modulation indices ranging between 0 and 1.
- $V_{dc}$  is the DC bus voltage of the inverter (Volts).
- $V_a$ ,  $V_b$ , and  $V_c$  are the output three-phase voltages (Volts).

## Ports

### Input

#### **D<sub>abc</sub>** — Duty cycle for three-phase voltage

scalar

Three-phase modulation indices in the range [0,1] for generating voltages that run the motor.

Data Types: single | double | fixed point

#### **V<sub>dc</sub>** — Inverter DC voltage

scalar

DC bus voltage input to the inverter.

Data Types: single | double | fixed point | uint8 | uint16 | uint32

## Output

### $V_{abc}$ — Three-phase voltage output

scalar

Three-phase voltage (Volts) corresponding to the input duty cycle that runs the motor.

Data Types: `single` | `double` | `fixed point`

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

[Space Vector Generator](#) | [Induction Motor](#) | [Interior PMSM](#) | [Surface Mount PMSM](#)

## Topics

“Open-Loop and Closed-Loop Control”

“Field-Oriented Control (FOC)”

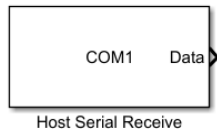
## Introduced in R2020a



# Host Serial Receive

Receive binary data over serial port

**Library:** Motor Control Blockset / Protection and Diagnostics



## Description

The Host Serial Receive block configures and opens an interface to the specified serial port. The configuration and initialization occur once at the start of the model's execution. The block acquires data from the serial port during the model's run time. You can use only one Host Serial Receive block at a time to receive data from a specific serial port.

---

**Note** You must configure your serial port parameters using the Host Serial Setup block before you specify the Host Serial Receive block parameters.

---

This block has no input ports. It has one or two output ports based on whether you select blocking or non-blocking mode. If you select blocking mode, the block has one output port, **Data**, corresponding to the data it receives. If you do not select blocking mode, the block has two output ports, **Data** and **Status**.

This block uses a First In, First Out (FIFO) buffer to receive data from the serial port. At each time step, the **Data** port returns the requested values from the buffer. In non-blocking mode, the **Status** port indicates if the block has received new data. If the **Status** port displays 1, new data is available and if the **Status** port displays 0, no new data is available.

## Other Supported Features

- The Host Serial Receive block supports the use of Simulink Accelerator mode, but not Rapid Accelerator. This feature speeds up the execution of Simulink models.
- The Host Serial Receive block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.
- The Host Serial Receive block supports C/C++ code generation. This feature allows you to generate C and C++ code using Simulink Coder™.

For more information on these features, see the “Simulink” documentation.

## Ports

### Output

#### Data — Data received

vector | matrix | array

Data received by the block from the serial port, returned as a vector, matrix, or array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

**Status — New data available**

`false` or `0` (default) | `true` or `1`

New data available status, returned as numeric or logical `1` (`true`) or `0` (`false`). If this port returns `1`, new data is available to be read.

**Dependencies**

To enable this port, unselect the **Enable blocking mode** parameter.

Data Types: `Boolean`

**Parameters****Port — Serial communication port**

available communication ports

Serial port on your machine that you want to receive data from. Select a port from the available ports and then configure the port using the Host Serial Setup block. If you have not configured a port, the block returns an error when you run your model.

---

**Note** Each Host Serial Receive block must have a configured Host Serial Setup block. If you use multiple serial ports in your simulation, you must configure each port using a separate Host Serial Setup block.

---

**Programmatic Use**

**Block Parameter:** `Port`

**Type:** character vector, string

**Data type — Output data type**

`uint8` (default) | `single` | `double` | `int8` | `int16` | `uint16` | `int32` | `uint32`

Data type that the block receives from the serial port, specified as a MATLAB numeric data type.

**Programmatic Use**

**Block Parameter:** `DataType`

**Type:** character vector, string

**Values:** `'uint8'` | `'single'` | `'double'` | `'int8'` | `'int16'` | `'uint16'` | `'int32'` | `'uint32'`

**Default:** `'uint8'`

**Header — Header**

numeric array | integer from 0 to 255, inclusive

If this parameter is selected, you can specify the header that indicates the beginning of your data block. The simulation disregards data that occurs before the header. The header data is not sent to the output port. By default, this parameter is not selected and no header is specified.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	"#"	uint8('#')	[35]
Numeric	"81"	uint8('81')	[56 49]
Alphabet	"Start"	uint8('Start')	[83 116 97 114 116]

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

#### Programmatic Use

**Block Parameter:** ToggleHeader

**Type:** character vector, string

**Values:** 'on' | 'off'

**Default:** 'off'

**Block Parameter:** Header

**Type:** character vector, string

**Values:** integer array

#### Terminator – Terminator

Custom Terminator (default) | <none> | CR ('\r') | LF ('\n') | CR/LF ('\r\n') | NULL ('\0')

If this parameter is selected, you can specify the terminator that indicates the end of your data block. The simulation considers any data that occurs after the terminator as a new data block. The terminator data is not sent to the output port. This terminator must match the terminator in the data you are reading from your serial port, if it has one.

If you select Custom Terminator, you can specify your own terminator value.

#### Programmatic Use

**Block Parameter:** ToggleTerminator

**Type:** character vector, string

**Values:** 'on' | 'off'

**Default:** 'off'

**Block Parameter:** Terminator

**Type:** character vector, string

**Values:** '<none>' | 'CR ('\r')' | 'LF ('\n')' | 'CR/LF ('\r\n')' | 'NULL ('\0')' | 'Custom Terminator'

**Default:** 'Custom Terminator'

#### Custom terminator – Custom terminator

numeric array | integer from 0 to 255, inclusive

Custom terminator that indicates the end of your data block. The simulation considers any data that occurs after the terminator as a new data block. The terminator data is not sent to the output port.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	"#"	uint8('#')	[35]
Numeric	"81"	uint8('81')	[56 49]
Alphabet	"End"	uint8('End')	[69 110 100]

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

#### Programmatic Use

**Block Parameter:** CustomTerminator

**Type:** character vector, string

**Values:** integer array

#### Input Format — Format of data read

Row major (default) | Column major

Format of the data that the block receives from the serial port, specified as Row major or Column major.

#### Programmatic Use

**Block Parameter:** InputFormat

**Type:** character vector, string

**Values:** 'Row major' | 'Column major'

**Default:** 'Row major'

#### Data size — Number of values read

[1 1] (default) | numeric array

Output data size, or the number of values that should be read at each simulation time step. This parameter is specified as a multidimensional numeric array. The data size does not include the header or terminator values.

#### Programmatic Use

**Block Parameter:** DataSize

**Type:** character vector, string

**Values:** integer array

**Default:** '[1 1]'

#### Enable blocking mode — Simulation waits while receiving data

on (default) | off

This parameter has the simulation wait while the block receives data. When new data becomes available, the simulation continues from the next time step. Unselect the check box if you do not want the read operation to cause the simulation to wait.

If you enable blocking mode, the simulation waits for the requested data to become available. The model waits for up to the amount of time specified by the **Timeout** parameter in the Host Serial Setup block. If new data does not become available during a simulation, you can return an error by selecting the Error option for the **Action when data is not available** parameter.

If you do not enable blocking mode, the simulation runs continuously and the block has two output ports, **Status** and **Data**. The **Data** port contains the requested set of data at each time step. The **Status** port contains 0 or 1 based on whether it received new data at the given time step.

**Programmatic Use****Block Parameter:** EnableBlockingMode**Type:** character vector, string**Values:** 'on' | 'off'**Default:** 'on'**Action when data is not available — Action to take when data not available**

Output last received value (default) | Output custom value | Error

Action the block should take when data is not available. Available options are:

- **Output last received value** — Block returns the value it received at the preceding time step when it does not receive data at current time step.
- **Output custom value** — Block returns any user-defined value when it does not receive current data. Define the custom value in the **Custom value** field.
- **Error** — Block returns an error when it does not receive current data. You must select **Enable blocking mode** to use this option.

**Programmatic Use****Block Parameter:** ActionDataUnavailable**Type:** character vector, string**Values:** 'Output last received value' | 'Output custom value' | 'Error'**Default:** 'Output last received value'**Custom value — Custom output value when data unavailable**

0 (default) | numeric

Custom value for the block to output when it does not receive new data. The custom value can be a scalar or value equal to the size of data that it receives (specified by **Data size** parameter). You must select **Output custom value** as the **Action when data is unavailable** to set this parameter.

**Programmatic Use****Block Parameter:** CustomValue**Type:** character vector, string**Values:** numeric**Default:** '0'**Block sample time — Sampling time**

-1 (default) | positive numeric

Sampling time of the block during the simulation. This is the rate at which the block is executed during simulation.

**Programmatic Use****Block Parameter:** SampleTime**Type:** character vector, string**Values:** positive numeric**Default:** '0.01'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block generates platform-specific code for the host machine's platform only (Windows®, macOS, Linux®).

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

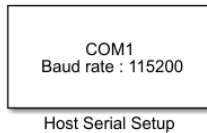
Host Serial Setup | Host Serial Transmit

**Introduced in R2020a**

# Host Serial Setup

Configure parameters for serial port

**Library:** Motor Control Blockset / Protection and Diagnostics



## Description

The Host Serial Setup block configures parameters for a serial port that you can use to send and receive data. Use this block to set the parameters of your serial port before you set up the Host Serial Receive and the Host Serial Transmit blocks.

---

**Note** You must configure your serial port parameters using the Host Serial Setup block before you specify the Host Serial Receive and Host Serial Transmit block parameters.

---

## Other Supported Features

- The Host Serial Setup block supports the use of Simulink Accelerator mode, but not Rapid Accelerator. This feature speeds up the execution of Simulink models.
- The Host Serial Setup block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.
- The Host Serial Setup block supports C/C++ code generation. This feature allows you to generate C and C++ code using Simulink Coder.

For more information on these features, see the “Simulink” documentation.

## Parameters

### Port — Serial communication port

available communication ports

Serial port on your machine that you want to configure. Use this configured port to send and receive data with your Host Serial Transmit and Host Serial Receive blocks. If you have not configured a port, the block returns an error when you run your model.

---

**Note** Each Host Serial Transmit and Host Serial Receive block must have a configured Host Serial Setup block. If you use multiple serial ports in your simulation, you must configure each port using a separate Host Serial Setup block.

---

### Programmatic Use

**Block Parameter:** Port

**Type:** character vector, string

**Baud rate — Communication speed**

115200 (default) | positive integer

Rate at which bits are transmitted for the serial interface, in bits per second.

**Programmatic Use****Block Parameter:** BaudRate**Type:** character vector, string**Values:** positive integer**Default:** '9600'**Data bits — Number of bits to represent one character of data**

8 (default) | 5 | 6 | 7

Number of data bits to transmit over the serial interface.

**Programmatic Use****Block Parameter:** DataBits**Type:** character vector, string**Values:** '5' | '6' | '7' | '8'**Default:** '8'**Parity — Parity bit type**

none (default) | even | odd

Parity bit type added to data transmitted by serial port. You can use this parameter to add a parity bit (also referred to as a check bit) to your data. Adding a parity bit to a string of binary code is a method of detecting errors in data transmission by ensuring that the total number of 1-bits is even or odd.

The value of the parity bit is determined by the number of 1s in a given set of bits and is set as follows.

Parity Bit Type	Parity Bit Value	
	If number of 1s is even	If number of 1s is odd
none	No parity bit set	No parity bit set
even	0	1
odd	1	0

**Programmatic Use****Block Parameter:** Parity**Type:** character vector, string**Values:** 'none' | 'even' | 'odd'**Default:** 'none'**Stop bits — Pattern of bits that indicates the end of a character**

1 (default) | positive scalar

Number of bits used to indicate the end of a byte. This parameter depends on the value you select for the **Data bits** parameter. If you select data bits 6, 7, or 8, the default value is 1 and the other available choice is 2. If you select data bit 5, the default value is 1 and the other available choice is 1.5.



**Programmatic Use****Block Parameter:** StopBits**Type:** character vector, string**Values:** positive scalar**Default:** '1'**Byte order — Sequential order of bytes**

little-endian (default) | big-endian

Sequential order in which bytes are arranged into larger numerical values. If the byte order is little-endian, then the instrument stores the first byte in the first memory address. If the byte order is big-endian, then the instrument stores the last byte in the first memory address.

Configure the byte order to the appropriate value for your instrument before performing a read or write operation. Refer to your instrument documentation for information about the order in which it stores bytes.

**Programmatic Use****Block Parameter:** ByteOrder**Type:** character vector, string**Values:** 'little-endian' | 'big-endian'**Default:** 'little-endian'**Flow control — Mode for managing data transmission rate**

none (default) | hardware

Process of managing the rate of data transmission on your serial port. Select none to have no flow control or hardware to let your hardware determine the flow control.

**Programmatic Use****Block Parameter:** FlowControl**Type:** character vector, string**Values:** 'none' | 'hardware'**Default:** none**Timeout — Allowed time to complete operations**

1.0 (default) | positive scalar

Amount of time that the model waits for data during each simulation time step.

**Programmatic Use****Block Parameter:** Timeout**Type:** character vector, string**Values:** positive scalar**Default:** '10'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block generates platform-specific code for the host machine's platform only (Windows, macOS, Linux).

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

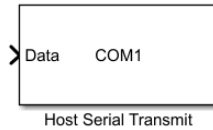
Host Serial Receive | Host Serial Transmit

**Introduced in R2020a**

# Host Serial Transmit

Send binary data over serial port

**Library:** Motor Control Blockset / Protection and Diagnostics



## Description

The Host Serial Transmit block configures and opens an interface to the specified serial port. The configuration and initialization occur once at the start of the model's execution. The block sends data from the model to the serial port during the model's run time. You can use multiple Host Serial Transmit blocks at a time to send data to a specific serial port.

---

**Note** You must configure your serial port parameters using the Host Serial Setup block before you specify the Host Serial Transmit block parameters.

---

The Host Serial Transmit block has one input port that accepts both 1-D vector and matrix data. This block has no output ports. The block inherits the data type from the signal at the input port. Valid data types are `single`, `double`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`.

## Other Supported Features

- The Host Serial Transmit block supports the use of Simulink Accelerator mode, but not Rapid Accelerator. This feature speeds up the execution of Simulink models.
- The Host Serial Transmit block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.
- The Host Serial Transmit block supports C/C++ code generation. This feature allows you to generate C and C++ code using Simulink Coder.

For more information on these features, see the “Simulink” documentation.

## Ports

### Input

#### Data — Data values to send

vector | matrix | array

Data values to send from the block over your serial port, specified as a vector, matrix, or array. Set the parameters for this block before you send data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

## Parameters

### Port — Serial communication port

available communication ports

Serial ports on your machine that you want to send data to. Select a port from the available ports and then configure the port using the Host Serial Setup block. If you have not configured a port, the block returns an error when you run your model.

---

**Note** Each Host Serial Transmit block must have a configured Host Serial Setup block. If you use multiple serial ports in your simulation, you must configure each port using a separate Host Serial Setup block.

---

### Programmatic Use

**Block Parameter:** Port

**Type:** character vector, string

### Header — Header

numeric array | integer from 0 to 255, inclusive

Header that indicates the beginning of your data block. The Host Serial Transmit block adds the header in front of the data before sending it over the serial port. By default, no header is specified.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	"#"	<code>uint8('#')</code>	[35]
Numeric	"81"	<code>uint8('81')</code>	[56 49]
Alphabet	"Start"	<code>uint8('Start')</code>	[83 116 97 114 116]

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

### Programmatic Use

**Block Parameter:** Header

**Type:** character vector, string

**Values:** integer array

### Terminator — Terminator

Custom Terminator (default) | <none> | CR ('\r') | LF ('\n') | CR/LF ('\r\n') | NULL ('\0')

Terminator that indicates the end of your data block. The Host Serial Transmit block appends the terminator to the data before sending it over the serial port.

If you select Custom Terminator, you can specify your own terminator value.

**Programmatic Use****Block Parameter:** Terminator**Type:** character vector, string**Values:** '<none>' | 'CR ('\r')' | 'LF ('\n')' | 'CR/LF ('\r\n')' | 'NULL ('\0')' | 'Custom Terminator'**Default:** 'Custom Terminator'**Custom terminator – Custom terminator**

numeric array | integer from 0 to 255, inclusive

Custom terminator that indicates the end of your data block. The Host Serial Transmit block appends the terminator to the data before sending it over the serial port.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	"#"	<code>uint8('#')</code>	[35]
Numeric	"81"	<code>uint8('81')</code>	[56 49]
Alphabet	"End"	<code>uint8('End')</code>	[69 110 100]

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

**Programmatic Use****Block Parameter:** CustomTerminator**Type:** character vector, string**Values:** integer array**Enable blocking mode – Simulation waits while sending data**

off (default) | on

This parameter has the simulation wait while the block sends data. Unselect the check box if you do not want the write operation to cause the simulation to wait.

If you enable blocking mode, the simulation waits for the data to be sent. If you do not enable blocking mode, the simulation runs continuously.

**Programmatic Use****Block Parameter:** EnableBlockingMode**Type:** character vector, string**Values:** 'on' | 'off'**Default:** 'off'**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

This block generates platform-specific code for the host machine's platform only (Windows, macOS, Linux).

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

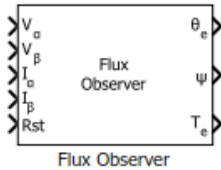
Host Serial Setup | Host Serial Receive

**Introduced in R2020a**

## Flux Observer

Compute electrical position, magnetic flux, and electrical torque of rotor

**Library:** Motor Control Blockset / Sensorless Estimators



### Description

The Flux Observer block computes the electrical position, magnetic flux, and electrical torque of a PMSM or an induction motor by using the per unit voltage and current values along the  $\alpha$ - and  $\beta$ -axes in the stationary  $\alpha\beta$  reference frame.

### Equations

These equations describe how the block computes the electrical position, magnetic flux, and electrical torque for a PMSM.

$$\psi_{\alpha} = \int (V_{\alpha} - I_{\alpha}R)dt - (L_s \cdot I_{\alpha})$$

$$\psi_{\beta} = \int (V_{\beta} - I_{\beta}R)dt - (L_s \cdot I_{\beta})$$

$$\psi = \sqrt{\psi_{\alpha}^2 + \psi_{\beta}^2}$$

$$T_e = \frac{3}{2}P(\psi_{\alpha}I_{\beta} - \psi_{\beta}I_{\alpha})$$

$$\theta_e = \tan^{-1} \frac{\psi_{\beta}}{\psi_{\alpha}}$$

These equations describe how the block computes the rotor electrical position, rotor magnetic flux, and electrical torque for an induction motor.

$$\psi_{\alpha} = \frac{L_r}{L_m} \left( \int (V_{\alpha} - I_{\alpha}R)dt - \sigma L_s I_{\alpha} \right)$$

$$\psi_{\beta} = \frac{L_r}{L_m} \left( \int (V_{\beta} - I_{\beta}R)dt - \sigma L_s I_{\beta} \right)$$

$$\sigma = 1 - \frac{L_m^2}{L_r \cdot L_s}$$

$$\psi = \sqrt{\psi_\alpha^2 + \psi_\beta^2}$$

$$T_e = \frac{3}{2} \cdot P \cdot \frac{L_m}{L_r} (\psi_\alpha I_\beta - \psi_\beta I_\alpha)$$

$$\theta_e = \tan^{-1} \frac{\psi_\beta}{\psi_\alpha}$$

where:

- $V_\alpha$  and  $V_\beta$  are the  $\alpha$ -axis and  $\beta$ -axis voltages (Volts).
- $I_\alpha$  and  $I_\beta$  are the  $\alpha$ -axis and  $\beta$ -axis current (Amperes).
- $R$  is the stator resistance of the motor (Ohms).
- $L_s$  is the stator inductance of the motor (Henry).
- $L_r$  is the rotor inductance of the motor (Henry).
- $L_m$  is the magnetizing inductance of the motor (Henry).
- $\sigma$  is the total leakage factor of the induction motor.
- $P$  is the number of motor pole pairs.
- $\psi$  is the rotor magnetic flux (Weber).
- $\psi_\alpha$  and  $\psi_\beta$  are the rotor magnetic fluxes along the  $\alpha$ - and  $\beta$ -axes (Weber).
- $T_e$  is the electrical torque of the rotor (Nm).
- $\theta_e$  is the electrical position of the rotor (Radians).

## Ports

### Input

#### $V_\alpha$ — $\alpha$ -axis voltage

scalar

Voltage component along the  $\alpha$ -axis in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### $V_\beta$ — $\beta$ -axis voltage

scalar

Voltage component along the  $\beta$ -axis in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### $I_\alpha$ — $\alpha$ -axis current

scalar

Current along the  $\alpha$ -axis in the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`



**$I_\beta$  —  $\beta$ -axis current**

scalar

Current along the  $\beta$ -axis in the stationary  $\alpha\beta$  reference frame.

Data Types: single | double | fixed point

**Rst — Reset block**

scalar

The pulse (true value) that resets the block algorithm.

Data Types: single | double | fixed point

**Output** **$\theta_e$  — Electrical position of motor**

scalar

The electrical position of the rotor as estimated by the block.

**Dependencies**

To enable this port, set **Block output** to Position.

Data Types: single | double | fixed point

 **$\Psi$  — Rotor flux of motor**

scalar

The magnetic flux of the rotor as estimated by the block.

**Dependencies**

To enable this port, set **Block output** to Flux.

Data Types: single | double | fixed point

 **$T_e$  — Electrical torque of motor**

scalar

The electrical torque of the rotor as estimated by the block.

**Dependencies**

To enable this port, set **Block output** to Torque.

Data Types: single | double | fixed point

**Parameters****Motor parameters****Motor selection — Type of motor**

PMSM (default) | ACIM

Select the type of motor that the block supports.

**Input units – Unit of voltage and current inputs**

SI unit (default) | Per-unit

Select the unit of the  $\alpha$  and  $\beta$ -axes voltage and current input values.

**Block output – Select outputs that block should compute**

Position (default) | Flux | Torque

Select one or more quantities that the block should compute and display in the block output.

---

**Note** You must select at least one value. The block displays an error message if you click **Ok** or **Apply** without selecting any value.

---

**Pole pairs – Number of pole pairs available in motor**

4 (default) | scalar

Number of pole pairs available in the motor.

**Dependencies**

To enable this parameter, set **Block output** to Torque.

**Stator resistance (ohm) – Stator winding resistance**

0.36 (default) | scalar

Stator phase winding resistance of the motor in ohms.

**Stator d-axis inductance (H) – Stator winding inductance along d-axis**

0.2e-3 (default) | scalar

Stator winding inductance of the motor along  $d$ -axis in Henry.

**Dependencies**

To enable this parameter, set **Motor selection** to PMSM.

**Stator leakage inductance (H) – Leakage inductance of stator winding**

0.0068 (default) | scalar

Leakage inductance of the induction motor stator winding in Henry.

**Dependencies**

To enable this parameter, set **Motor selection** to ACIM.

**Rotor leakage inductance (H) – Leakage inductance of rotor winding**

0.0068 (default) | scalar

Leakage inductance of the induction motor rotor winding in Henry.

**Dependencies**

To enable this parameter, set **Motor selection** to ACIM.

**Magnetizing inductance (H) – Magnetizing inductance of induction motor**

0.0300 (default) | scalar

Magnetizing inductance of the induction motor in Henry.

#### Dependencies

To enable this parameter, set **Motor selection** to ACIM.

#### Cutoff frequency (Hz) – Cutoff frequency of internal high-pass filter

3.1863 (default) | scalar

Cutoff frequency of the internal high-pass filter (that filters noise) in Hertz.

The Flux Observer block uses an internal first order IIR high-pass filter. You should set the **Cutoff frequency (Hz)** for this filter to a value that is lower than the lowest frequency corresponding to the minimum speed of the motor. For example, you can enter a value that is one-tenth of the lowest electrical frequency of the stator voltages and the currents. However, you can adjust this value to determine a more accurate cutoff frequency that generates the desired block output.

#### Discrete step size (s) – Sample time after which block executes again

50e-6 (default) | scalar

The fixed time interval in seconds between two consecutive instances of block execution.

#### Datatypes

##### Position unit – Unit of electrical position output

Radians (default) | Degrees | Per-unit

Unit of the electrical position output.

#### Dependencies

To enable this parameter, set **Block output** to Position.

##### Position datatype – Data type of electrical position output

single (default) | double | fixed point

Data type of the electrical position output.

#### Dependencies

To enable this parameter, set **Block output** to Position.

##### Flux unit – Unit of magnetic flux output

Weber (default) | Per-unit

Unit of the magnetic flux output.

#### Dependencies

To enable this parameter, set **Block output** to Flux.

##### Flux datatype – Data type of magnetic flux output

single (default) | double | fixed point

Data type of the magnetic flux output.

**Dependencies**

To enable this parameter, set **Block output** to Flux.

**Torque unit – Unit of electrical torque output**

Nm (default) | Per-unit

Unit of the electrical torque output.

**Dependencies**

To enable this parameter, set **Block output** to Torque.

**Torque datatype – Data type of electrical torque output**

single (default) | double | fixed point

Data type of the electrical torque output.

**Dependencies**

To enable this parameter, set **Block output** to Torque.

**References**

- [1] O. Sandre-Hernandez, J. J. Rangel-Magdaleno and R. Morales-Caporal, "Simulink-HDL cosimulation of direct torque control of a PM synchronous machine based FPGA," *2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), Campeche, 2014*, pp. 1-6. (doi: 10.1109/ICEEE.2014.6978298)
- [2] Y. Inoue, S. Morimoto and M. Sanada, "Control method suitable for direct torque control based motor drive system satisfying voltage and current limitations," *The 2010 International Power Electronics Conference - ECCE ASIA -, Sapporo, 2010*, pp. 3000-3006. (doi: 10.1109/IPEC.2010.5543698)

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

Sliding Mode Observer | Clarke Transform | Inverse Park Transform | Speed Measurement

**Topics**

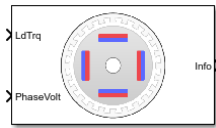
"Open-Loop and Closed-Loop Control"  
"Field-Oriented Control (FOC)"

**Introduced in R2020a**

## Interior PMSM

Three-phase interior permanent magnet synchronous motor with sinusoidal back electromotive force

**Library:** Powertrain Blockset / Propulsion / Electric Motors and Inverters  
Motor Control Blockset / Electrical Systems / Motors



### Description

The Interior PMSM block implements a three-phase interior permanent magnet synchronous motor (PMSM) with sinusoidal back electromotive force. The block uses the three-phase input voltages to regulate the individual phase currents, allowing control of the motor torque or speed.

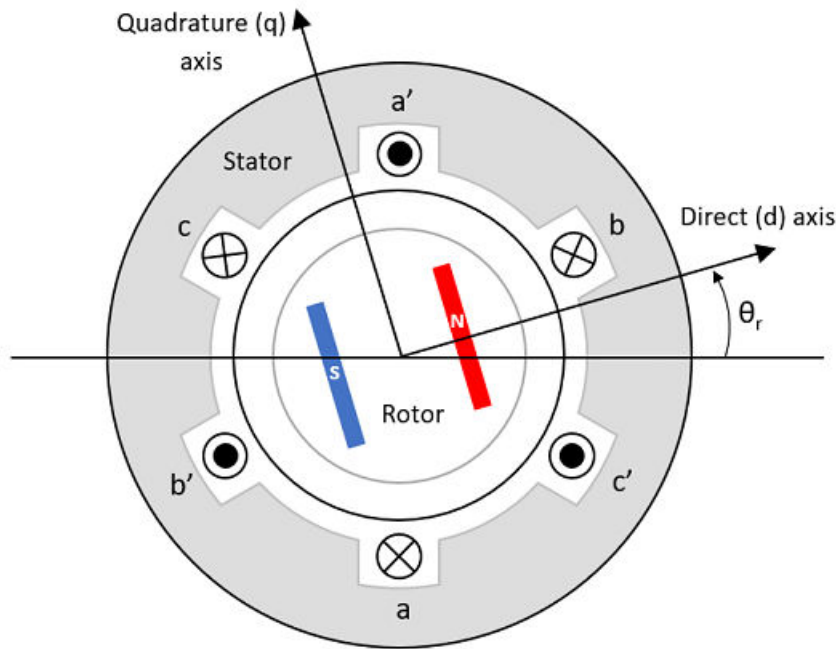
By default, the block sets the **Simulation type** parameter to **Continuous** to use a continuous sample time during simulation. If you want to generate code for fixed-step double- and single-precision targets, considering setting the parameter to **Discrete**. Then specify a **Sample Time, Ts** parameter.

On the **Parameters** tab, if you select **Back-emf**, the block implements this equation to calculate the permanent flux linkage constant.

$$\lambda_{pm} = \frac{1}{\sqrt{3}} \cdot \frac{K_e}{1000P} \cdot \frac{60}{2\pi}$$

### Motor Construction

This figure shows the motor construction with a single pole pair on the motor.



The motor magnetic field due to the permanent magnets creates a sinusoidal rate of change of flux with motor angle.

For the axes convention, the *a*-phase and permanent magnet fluxes are aligned when motor angle  $\theta_r$  is zero.

### Three-Phase Sinusoidal Model Electrical System

The block implements these equations, expressed in the motor flux reference frame (dq frame). All quantities in the motor reference frame are referred to the stator.

$$\omega_e = P\omega_m$$

$$\frac{d}{dt}i_d = \frac{1}{L_d}v_d - \frac{R}{L_d}i_d + \frac{L_q}{L_d}P\omega_m i_q$$

$$\frac{d}{dt}i_q = \frac{1}{L_q}v_q - \frac{R}{L_q}i_q - \frac{L_d}{L_q}P\omega_m i_d - \frac{\lambda_{pm}P\omega_m}{L_q}$$

$$T_e = 1.5P[\lambda_{pm}i_q + (L_d - L_q)i_d i_q]$$

The  $L_q$  and  $L_d$  inductances represent the relation between the phase inductance and the motor position due to the saliency of the motor.

The equations use these variables.

$L_q, L_d$	q- and d-axis inductances (H)
$R$	Resistance of the stator windings (ohm)
$i_q, i_d$	q- and d-axis currents (A)
$v_q, v_d$	q- and d-axis voltages (V)

$\omega_m$	Angular mechanical velocity of the motor (rad/s)
$\omega_e$	Angular electrical velocity of the motor (rad/s)
$\lambda_{pm}$	Permanent flux linkage constant (Wb)
$K_e$	Back electromotive force (EMF) (Vpk_LL/krpm, where Vpk_LL is the peak voltage line-to-line measurement)
$P$	Number of pole pairs
$T_e$	Electromagnetic torque (Nm)
$\theta_e$	Electrical angle (rad)

### Mechanical System

The motor angular velocity is given by:

$$\frac{d}{dt}\omega_m = \frac{1}{J}(T_e - T_f - F\omega_m - T_m)$$

$$\frac{d\theta_m}{dt} = \omega_m$$

The equations use these variables.

$J$	Combined inertia of motor and load (kgm <sup>2</sup> )
$F$	Combined viscous friction of motor and load (N·m/(rad/s))
$\theta_m$	Motor mechanical angular position (rad)
$T_m$	Motor shaft torque (Nm)
$T_e$	Electromagnetic torque (Nm)
$T_f$	Motor shaft static friction torque (Nm)
$\omega_m$	Angular mechanical velocity of the motor (rad/s)

### Power Accounting

For the power accounting, the block implements these equations.

Bus Signal		Description	Variable	Equations
PwrIn fo	PwrTrnsfrd — Power transferred between blocks	PwrMtr	Mechanical power	$P_{mot} = -\omega_m T_e$
	<ul style="list-style-type: none"> <li>Positive signals indicate flow into block</li> <li>Negative signals indicate flow out of block</li> </ul>	PwrBus	Electrical power	$P_{bus} = v_{an}i_a + v_{bn}i_b + v_{cn}i_c$
	PwrNotTrnsfrd — Power crossing the block boundary, but not transferred	PwrElecLoss	Resistive power loss	$P_{elec} = -\frac{3}{2}(R_{s1}i_{sd}^2 + R_{s1}i_{sq}^2)$
	<ul style="list-style-type: none"> <li>Positive signals indicate an input</li> </ul>			

Bus Signal		Description	Variable	Equations
	<ul style="list-style-type: none"> <li>Negative signals indicate a loss</li> </ul>	PwrMech Loss	Mechanical power loss	$P_{mech}$ When <b>Port Configuration</b> is set to Torque: $P_{mech} = -(\omega_m^2 F +  \omega_m  T_f)$ When <b>Port Configuration</b> is set to Speed: $P_{mech} = 0$
	PwrStored — Stored energy rate of change <ul style="list-style-type: none"> <li>Positive signals indicate an increase</li> <li>Negative signals indicate a decrease</li> </ul>	PwrMtrStored	Stored motor power	$P_{str} = \frac{P_{bus}}{P_{elec}} + \frac{P_{mot}}{P_{mech}}$

The equations use these variables.

- $R_s$  Stator resistance (ohm)
- $i_a, i_b, i_c$  Stator phase a, b, and c current (A)
- $i_{sq}, i_{sd}$  Stator q- and d-axis currents (A)
- $v_{an}, v_{bn}, v_{cn}$  Stator phase a, b, and c voltage (V)
- $\omega_m$  Angular mechanical velocity of the rotor (rad/s)
- $F$  Combined motor and load viscous damping (N·m/(rad/s))
- $T_e$  Electromagnetic torque (Nm)
- $T_f$  Combined motor and load friction torque (Nm)

### Amplitude invariant dq transformation

The block uses these equations to implement amplitude invariant  $dq$  transformation to ensure that the  $dq$  and three phase amplitudes are equal.

$$\begin{bmatrix} v_{sd} \\ v_{sq} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta_{da}) & \cos(\theta_{da} - \frac{2\pi}{3}) & \cos(\theta_{da} + \frac{2\pi}{3}) \\ -\sin(\theta_{da}) & -\sin(\theta_{da} - \frac{2\pi}{3}) & -\sin(\theta_{da} + \frac{2\pi}{3}) \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}$$

$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \begin{bmatrix} \cos(\theta_{da}) & -\sin(\theta_{da}) \\ \cos(\theta_{da} - \frac{2\pi}{3}) & -\sin(\theta_{da} - \frac{2\pi}{3}) \\ \cos(\theta_{da} + \frac{2\pi}{3}) & -\sin(\theta_{da} + \frac{2\pi}{3}) \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix}$$

The equations use these variables.



$\theta_{da}$	$dq$ stator electrical angle with respect to the rotor $a$ -axis (rad)
$v_{sq}, v_{sd}$	Stator $q$ - and $d$ -axis voltages (V)
$i_{sq}, i_{sd}$	Stator $q$ - and $d$ -axis currents (A)
$v_a, v_b, v_c$	Stator voltage phases $a, b, c$ (V)
$i_a, i_b, i_c$	Stator currents phases $a, b, c$ (A)

## Ports

### Input

#### LdTrq — Load torque on motor

scalar

Load torque on the motor shaft,  $T_m$ , in N·m.

#### Dependencies

To create this port, select Torque for the **Port Configuration** parameter.

#### Spd — Motor shaft speed

scalar

Angular velocity of the motor,  $\omega_m$ , in rad/s.

#### Dependencies

To create this port, select Speed for the **Port Configuration** parameter.

#### PhaseVolt — Stator terminal voltages

1-by-3 array

Stator terminal voltages,  $V_a, V_b,$  and  $V_c$ , in V.

#### Dependencies

To create this port, select Speed or Torque for the **Port Configuration** parameter.

### Output

#### Info — Bus signal

bus

The bus signal contains these block calculations.

Signal	Description	Variable	Units
IaStator	Stator phase current A	$i_a$	A
IbStator	Stator phase current B	$i_b$	A
IcStator	Stator phase current C	$i_c$	A
IdSync	Direct axis current	$i_d$	A
IqSync	Quadrature axis current	$i_q$	A
VdSync	Direct axis voltage	$v_d$	V

Signal		Description	Variable	Units	
VqSync		Quadrature axis voltage	$v_q$	V	
MtrSpd		Angular mechanical velocity of the motor	$\omega_m$	rad/s	
MtrPos		Motor mechanical angular position	$\theta_m$	rad	
MtrTrq		Electromagnetic torque	$T_e$	N·m	
PwrInfo	PwrTrnsfrd	PwrMtr	Mechanical power	$P_{mot}$	W
		PwrBus	Electrical power	$P_{bus}$	W
	PwrNotTrnsfrd	PwrElecLoss	Resistive power loss	$P_{elec}$	W
		PwrMechLoss	Mechanical power loss	$P_{mech}$	W
	PwrStored	PwrMtrStored	Stored motor power	$P_{str}$	W

**PhaseCurr — Phase a, b, c current**

1-by-3 array

Phase a, b, c current,  $i_a$ ,  $i_b$ , and  $i_c$ , in A.

**MtrTrq — Motor torque**

scalar

Motor torque,  $T_{mtr}$ , in N·m.

**Dependencies**

To create this port, select Speed for the **Mechanical input configuration** parameter.

**MtrSpd — Motor speed**

scalar

Angular speed of the motor,  $\omega_{mtr}$ , in rad/s.

**Dependencies**

To create this port, select Torque for the **Mechanical input configuration** parameter.

**Parameters**

**Block Options**

**Mechanical input configuration — Select port configuration**

Torque (default) | Speed

This table summarizes the port configurations.

Port Configuration	Creates Input Port	Creates Output Port
Torque	LdTrq	MtrSpd

Port Configuration	Creates Input Port	Creates Output Port
Speed	Spd	MtrTrq

### Simulation type – Select simulation type

Continuous (default) | Discrete

By default, the block uses a continuous sample time during simulation. If you want to generate code for single-precision targets, considering setting the parameter to **Discrete**.

#### Dependencies

Setting **Simulation type** to **Discrete** creates the **Sample Time, Ts** parameter.

### Sample Time (Ts) – Sample time for discrete integration

scalar

Integration sample time for discrete simulation, in s.

#### Dependencies

Setting **Simulation type** to **Discrete** creates the **Sample Time, Ts** parameter.

#### Load Parameters

### File – Path to motor parameter ".m" or ".mat" file

scalar

Enter the path to the motor parameter ".m" or ".mat" file that you saved using the Motor Control Blockset parameter estimation tool. You can also click the **Browse** button to navigate and select the ".m" or ".mat" file, and update **File** parameter with the file name and path. For details related to the motor parameter estimation process, see "Estimate PMSM Parameters Using Recommended Hardware".

- **Load from file** - Click this button to read the estimated motor parameters from the ".m" or ".mat" file (indicated by the **File** parameter) and load them to the motor block.
- **Save to file** - Click this button to read the motor parameters from the motor block and save them into a ".m" or ".mat" file (with a file name and location that you specify in the **File** parameter).

---

**Note** Before you click **Save to file** button, ensure that the target file name in the **File** parameter has either ".m" or ".mat" extension. If you use any other file extension, the block displays an error message.

---

#### Parameters

### Number of pole pairs (P) – Pole pairs

scalar

Motor pole pairs,  $P$ .

### Stator phase resistance per phase (Rs) – Resistance

scalar

Stator phase resistance per phase,  $R_s$ , in ohm.

**Stator d-axis and q-axis inductance (Ldq) – Inductance**  
vector

Stator d-axis and q-axis inductance,  $L_d$ ,  $L_q$ , in H.

**Permanent flux linkage constant (lambda\_pm) – Flux**  
scalar

Permanent flux linkage constant,  $\lambda_{pm}$ , in Wb.

**Back-emf constant (Ke) – Back electromotive force**  
scalar

Back electromotive force, EMF,  $K_e$ , in Vpk\_LL/krpm. Vpk\_LL is the peak voltage line-to-line measurement.

To calculate the permanent flux linkage constant, the block implements this equation.

$$\lambda_{pm} = \frac{1}{\sqrt{3}} \cdot \frac{K_e}{1000P} \cdot \frac{60}{2\pi}$$

**Physical inertia, viscous damping, and static friction (mechanical) – Inertia, damping, friction**  
vector

Mechanical properties of the motor:

- Inertia,  $J$ , in kg.m<sup>2</sup>
- Viscous damping,  $F$ , in N·m/(rad/s)
- Static friction,  $T_f$ , in N·m

**Dependencies**

To enable this parameter, select the Torque configuration parameter.

**Initial Values****Initial d-axis and q-axis current (idq0) – Current**  
vector

Initial q- and d-axis currents,  $i_q$ ,  $i_d$ , in A.

**Initial mechanical position (theta\_init) – Angle**  
scalar

Initial motor angular position,  $\theta_{m0}$ , in rad.

**Initial mechanical speed (omega\_init) – Speed**  
scalar

Initial angular velocity of the motor,  $\omega_{m0}$ , in rad/s.

**Dependencies**

To enable this parameter, select the Torque configuration parameter.

## References

- [1] Kundur, P. *Power System Stability and Control*. New York, NY: McGraw Hill, 1993.
- [2] Anderson, P. M. *Analysis of Faulted Power Systems*. Hoboken, NJ: Wiley-IEEE Press, 1995.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

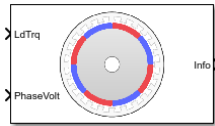
## See Also

**Introduced in R2017a**

## Surface Mount PMSM

Three-phase exterior permanent magnet synchronous motor with sinusoidal back electromotive force

**Library:** Powertrain Blockset / Propulsion / Electric Motors and Inverters  
Motor Control Blockset / Electrical Systems / Motors



### Description

The Surface Mount PMSM block implements a three-phase exterior permanent magnet synchronous motor (PMSM) with sinusoidal back electromotive force. The block uses the three-phase input voltages to regulate the individual phase currents, allowing the control of the motor torque or speed.

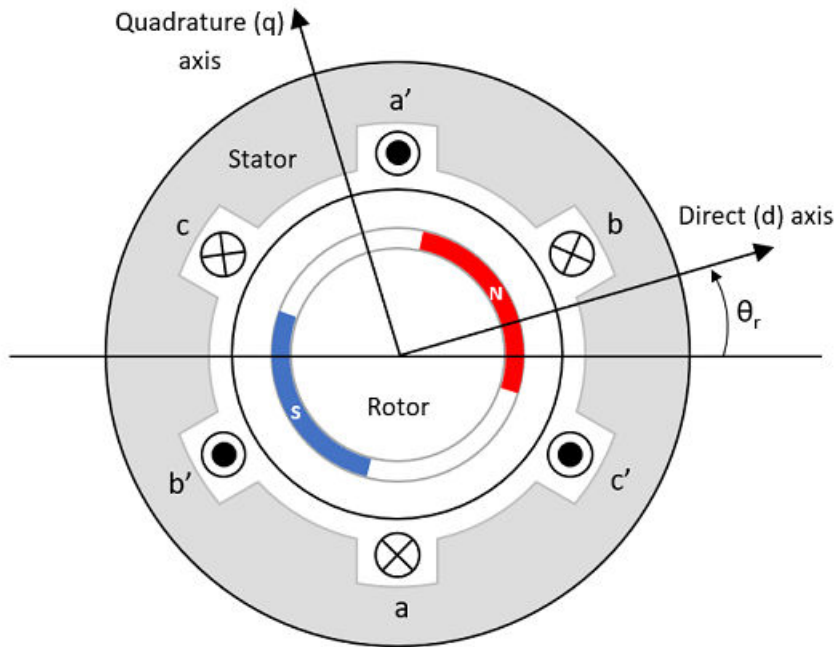
By default, the block sets the **Simulation type** parameter to **Continuous** to use a continuous sample time during simulation. If you want to generate code for fixed-step double- and single-precision targets, considering setting the parameter to **Discrete**. Then specify a **Sample Time, Ts** parameter.

On the **Parameters** tab, if you select **Back-emf** or **Torque constant**, the block implements one of these equations to calculate the permanent flux linkage constant.

Setting	Equation
Back-emf	$\lambda_{pm} = \frac{1}{\sqrt{3}} \cdot \frac{K_e}{1000P} \cdot \frac{60}{2\pi}$
Torque constant	$\lambda_{pm} = \frac{2}{3} \cdot \frac{K_t}{P}$

### Motor Construction

This figure shows the motor construction with a single pole pair on the motor.



The motor magnetic field due to the permanent magnets creates a sinusoidal rate of change of flux with motor angle.

For the axes convention, the  $a$ -phase and permanent magnet fluxes are aligned when motor angle  $\theta_r$  is zero.

### Three-Phase Sinusoidal Model Electrical System

The block implements these equations, expressed in the motor flux reference frame (dq frame). All quantities in the motor reference frame are referred to the stator.

$$\omega_e = P\omega_m$$

$$\frac{d}{dt}i_d = \frac{1}{L_d}v_d - \frac{R}{L_d}i_d + \frac{L_q}{L_d}P\omega_m i_q$$

$$\frac{d}{dt}i_q = \frac{1}{L_q}v_q - \frac{R}{L_q}i_q - \frac{L_d}{L_q}P\omega_m i_d - \frac{\lambda_{pm}P\omega_m}{L_q}$$

$$T_e = 1.5P[\lambda_{pm}i_q + (L_d - L_q)i_d i_q]$$

The  $L_q$  and  $L_d$  inductances represent the relation between the phase inductance and the motor position due to the saliency of the motor magnets. For the surface mount PMSM,  $L_d = L_q$ .

The equations use these variables.

$L_q, L_d$	q- and d-axis inductances (H)
$R$	Resistance of the stator windings (ohm)
$i_q, i_d$	q- and d-axis currents (A)
$v_q, v_d$	q- and d-axis voltages (V)

$\omega_m$	Angular mechanical velocity of the motor (rad/s)
$\omega_e$	Angular electrical velocity of the motor (rad/s)
$\lambda_{pm}$	Permanent magnet flux linkage (Wb)
$K_e$	Back electromotive force (EMF) (Vpk_LL/krpm, where Vpk_LL is the peak voltage line-to-line measurement)
$K_t$	Torque constant (N·m/A)
$P$	Number of pole pairs
$T_e$	Electromagnetic torque (Nm)
$\Theta_e$	Electrical angle (rad)

### Mechanical System

The motor angular velocity is given by:

$$\frac{d}{dt}\omega_m = \frac{1}{J}(T_e - T_f - F\omega_m - T_m)$$

$$\frac{d\theta_m}{dt} = \omega_m$$

The equations use these variables.

$J$	Combined inertia of motor and load (kgm <sup>2</sup> )
$F$	Combined viscous friction of motor and load (N·m/(rad/s))
$\theta_m$	Motor mechanical angular position (rad)
$T_m$	Motor shaft torque (Nm)
$T_e$	Electromagnetic torque (Nm)
$T_f$	Motor shaft static friction torque (Nm)
$\omega_m$	Angular mechanical velocity of the motor (rad/s)

### Power Accounting

For the power accounting, the block implements these equations.

Bus Signal			Description	Variable	Equations
PwrInfo	PwrTrnsfrd — Power transferred between blocks	PwrMtr	Mechanical power	$P_{mot}$	$P_{mot} = -\omega_m T_e$
	<ul style="list-style-type: none"> <li>Positive signals indicate flow into block</li> <li>Negative signals indicate flow out of block</li> </ul>	PwrBus	Electrical power	$P_{bus}$	$P_{bus} = v_{an}i_a + v_{bn}i_b + v_{cn}i_c$
	PwrNotTrnsfrd — Power crossing the block boundary, but not transferred	PwrElecLoss	Resistive power loss	$P_{elec}$	$P_{elec} = -\frac{3}{2}(R_s i_{sd}^2 + R_s i_{sq}^2)$



Bus Signal		Description	Variable	Equations
<ul style="list-style-type: none"> <li>Positive signals indicate an input</li> <li>Negative signals indicate a loss</li> </ul>	PwrMech Loss	Mechanical power loss	$P_{mech}$	When <b>Port Configuration</b> is set to Torque:  $P_{mech} = -(\omega_m^2 F +  \omega_m  T_f)$ When <b>Port Configuration</b> is set to Speed:  $P_{mech} = 0$
PwrStored — Stored energy rate of change  <ul style="list-style-type: none"> <li>Positive signals indicate an increase</li> <li>Negative signals indicate a decrease</li> </ul>	PwrMtrStored	Stored motor power	$P_{str}$	$P_{str} = \frac{P_{bus} + P_{mot} + P_{elec}}{P_{mech}}$

The equations use these variables.

$R_s$	Stator resistance (ohm)
$i_a, i_b, i_c$	Stator phase a, b, and c current (A)
$i_{sq}, i_{sd}$	Stator q- and d-axis currents (A)
$v_{an}, v_{bn}, v_{cn}$	Stator phase a, b, and c voltage (V)
$\omega_m$	Angular mechanical velocity of the motor (rad/s)
$F$	Combined motor and load viscous damping N·m/(rad/s)
$T_e$	Electromagnetic torque (Nm)
$T_f$	Combined motor and load friction torque (Nm)

## Ports

### Input

#### LdTrq — Load torque on motor

scalar

Load torque on the motor shaft,  $T_m$ , in N·m.

#### Dependencies

To create this port, select Torque for the **Port Configuration** parameter.

#### Spd — Motor shaft speed

scalar

Angular velocity of the motor,  $\omega_m$ , in rad/s.

**Dependencies**

To create this port, select Speed for the **Port Configuration** parameter.

**PhaseVolt – Stator terminal voltages**

1-by-3 array

Stator terminal voltages,  $V_a$ ,  $V_b$ , and  $V_c$ , in V.

**Output**

**Info – Bus signal**

bus

The bus signal contains these block calculations.

Signal			Description	Variable	Units
IaStator			Stator phase current A	$i_a$	A
IbStator			Stator phase current B	$i_b$	A
IcStator			Stator phase current C	$i_c$	A
IdSync			Direct axis current	$i_d$	A
IqSync			Quadrature axis current	$i_q$	A
VdSync			Direct axis voltage	$v_d$	V
VqSync			Quadrature axis voltage	$v_q$	V
MtrSpd			Angular mechanical velocity of the motor	$\omega_m$	rad/s
MtrPos			Motor mechanical angular position	$\theta_m$	rad
MtrTrq			Electromagnetic torque	$T_e$	N·m
PwrInfo	PwrTrnsfrd	PwrMtr	Mechanical power	$P_{mot}$	W
		PwrBus	Electrical power	$P_{bus}$	W
	PwrNotTrnsfrd	PwrElecLoss	Resistive power loss	$P_{elec}$	W
		PwrMechLoss	Mechanical power loss	$P_{mech}$	W
	PwrStored	PwrMtrStored	Stored motor power	$P_{str}$	W

**PhaseCurr – Phase a, b, c current**

1-by-3 array

Phase a, b, c current,  $i_a$ ,  $i_b$ , and  $i_c$ , in A.

**MtrTrq – Motor torque**

scalar

Motor torque,  $T_{mtr}$ , in N·m.

**Dependencies**

To create this port, select Speed for the **Mechanical input configuration** parameter.

**MtrSpd — Motor speed**

scalar

Angular speed of the motor,  $\omega_{mtr}$  in rad/s.

**Dependencies**

To create this port, select Torque for the **Mechanical input configuration** parameter.

**Parameters****Block Options****Mechanical input configuration — Select port configuration**

Torque (default) | Speed

This table summarizes the port configurations.

Port Configuration	Creates Input Port	Creates Output Port
Torque	LdTrq	MtrSpd
Speed	Spd	MtrTrq

**Simulation type — Select simulation type**

Continuous (default) | Discrete

By default, the block uses a continuous sample time during simulation. If you want to generate code for single-precision targets, considering setting the parameter to **Discrete**.

**Dependencies**

Setting **Simulation type** to **Discrete** creates the **Sample Time, Ts** parameter.

**Sample Time (Ts) — Sample time for discrete integration**

scalar

Integration sample time for discrete simulation, in s.

**Dependencies**

Setting **Simulation type** to **Discrete** creates the **Sample Time, Ts** parameter.

**Load Parameters****File — Path to motor parameter ".m" or ".mat" file**

scalar

Enter the path to the motor parameter ".m" or ".mat" file that you saved using the Motor Control Blockset parameter estimation tool. You can also click the **Browse** button to navigate and select the ".m" or ".mat" file, and update **File** parameter with the file name and path. For details related to the motor parameter estimation process, see "Estimate PMSM Parameters Using Recommended Hardware".

- **Load from file** - Click this button to read the estimated motor parameters from the ".m" or ".mat" file (indicated by the **File** parameter) and load them to the motor block.
- **Save to file** - Click this button to read the motor parameters from the motor block and save them into a ".m" or ".mat" file (with a file name and location that you specify in the **File** parameter).

---

**Note** Before you click **Save to file** button, ensure that the target file name in the **File** parameter has either ".m" or ".mat" extension. If you use any other file extension, the block displays an error message.

---

### Parameters

#### Number of pole pairs (P) – Pole pairs

scalar

Motor pole pairs,  $P$ .

#### Stator phase resistance per phase (Rs) – Resistance

scalar

Stator phase resistance per phase,  $R_s$ , in ohm.

#### Stator d-axis inductance (Ldq\_) – Inductance

scalar

Stator inductance,  $L_{dq}$ , in H.

#### Permanent flux linkage constant (lambda\_pm) – Flux

scalar

Permanent flux linkage constant,  $\lambda_{pm}$ , in Wb.

#### Back-emf constant (Ke) – Back electromotive force

scalar

Back electromotive force, EMF,  $K_e$ , in peak Vpk\_LL/krpm. Vpk\_LL is the peak voltage line-to-line measurement.

To calculate the permanent flux linkage constant, the block implements this equation.

$$\lambda_{pm} = \frac{1}{\sqrt{3}} \cdot \frac{K_e}{1000P} \cdot \frac{60}{2\pi}$$

#### Torque constant (Kt) – Torque constant

scalar

Torque constant,  $K_t$ , in N·m/A.

To calculate the permanent flux linkage constant, the block implements this equation.

$$\lambda_{pm} = \frac{2}{3} \cdot \frac{K_t}{P}$$

## Physical inertia, viscous damping, and static friction (mechanical) — Inertia, damping, friction

vector

Mechanical properties of the motor:

- Inertia,  $J$ , in  $\text{kg}\cdot\text{m}^2$
- Viscous damping,  $F$ , in  $\text{N}\cdot\text{m}/(\text{rad}/\text{s})$
- Static friction,  $T_f$ , in  $\text{N}\cdot\text{m}$

### Dependencies

To enable this parameter, select the Torque configuration parameter.

### Initial Values

#### Initial d-axis and q-axis current (idq0) — Current

vector

Initial q- and d-axis currents,  $i_q$ ,  $i_d$ , in A.

#### Initial mechanical position (theta\_init) — Angle

scalar

Initial motor angular position,  $\theta_{m0}$ , in rad.

#### Initial mechanical speed (omega\_init) — Speed

scalar

Initial angular velocity of the motor,  $\omega_{m0}$ , in rad/s.

### Dependencies

To enable this parameter, select the Torque configuration parameter.

## References

- [1] Kundur, P. *Power System Stability and Control*. New York, NY: McGraw Hill, 1993.
- [2] Anderson, P. M. *Analysis of Faulted Power Systems*. Hoboken, NJ: Wiley-IEEE Press, 1995.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

## See Also

Introduced in R2017a

# Field Oriented Control Autotuner

Automatically and sequentially tune multiple PID control loops in field-oriented control application

**Library:** Motor Control Blockset / Controls / Controllers



## Description

The Field Oriented Control Autotuner block allows you to automatically tune PID control loops in your field-oriented control (FOC) application in real time. For more information on field-oriented control, see “Field-Oriented Control (FOC)”.

You can automatically tune PID controllers associated with the following loops:

- Direct-axis (d-axis) current loop
- Quadrature-axis (q-axis) current loop
- Speed loop
- Flux loop

For each loop the block tunes, the Field Oriented Control Autotuner block performs the autotuning experiment in closed-loop without a parametric model associated with that loop. The block allows you to specify the order in which the control loops are tuned. When the tuning experiment is running for one loop, the block has no effect on the other loops. During the experiment, the block:

- 1 Injects a test signal into the plant associated with that loop to collect plant input-output data and estimate frequency response in real time. The test signal is combination of sinusoidal perturbation signals added on top of the plant input.
- 2 At the end of the experiment, tunes PID controller parameters based on estimated plant frequency responses near the target bandwidth.
- 3 Writes updated PID gains at the block output, allowing you to transfer the new gains to existing controllers and validate the closed-loop performance.

You can use the Field Oriented Control Autotuner block to tune the existing PID controllers in your FOC structure. If you do not have the initial PID controllers, you can use the “Estimate Control Gains and Use Utility Functions” workflow to obtain them. You can then use the Field Oriented Control Autotuner block for refinement or retuning.

If you have a code-generation product such as Simulink Coder, you can generate code that implements the tuning algorithm on hardware, letting you tune in real time, using or without using Simulink to manage the autotuning process.

If you have a machine modeled in Simulink with Motor Control Blockset and an initial FOC structure with PID controllers, you can perform closed-loop PID autotuning against the modeled machine. Doing so lets you preview the plant response and adjust the settings for PID autotuning before tuning the controller in real time.

The block supports code generation with Simulink Coder, Embedded Coder®, and Simulink PLC Coder™. It does not support code generation with HDL Coder. For real-time applications, deploy the generated code on a rapid prototyping hardware such as Speedgoat® real-time target machine.

For more information about using the Field Oriented Control Autotuner block, see “How to Use Field Oriented Control Autotuner Block”.

This block requires Simulink Control Design software.

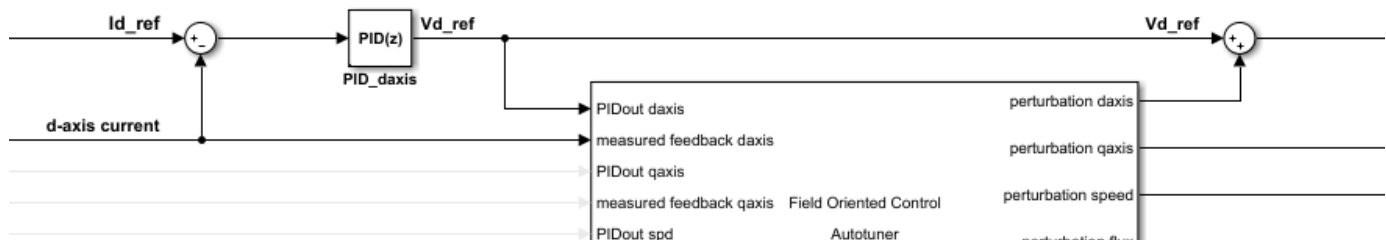
## Ports

### Input

#### **PIDout\_daxis** — Signal from direct-axis current controller

scalar

This port accepts the output of the d-axis controller `PID_daxis`, which is the output of PID controller that regulates the d-axis current of the motor. The controller generates the d-axis voltage reference `Vd_ref`, while the FOC autotuner block generates perturbations used during the tuning experiment for the d-axis current loop.



### Dependencies

To enable this port, select **Tune D-axis current loop**.

Data Types: `single` | `double`

#### **measured feedback\_daxis** — Measured direct-axis current

scalar

This port accepts the d-axis current obtained from the measured (sensed or estimated) motor currents.

### Dependencies

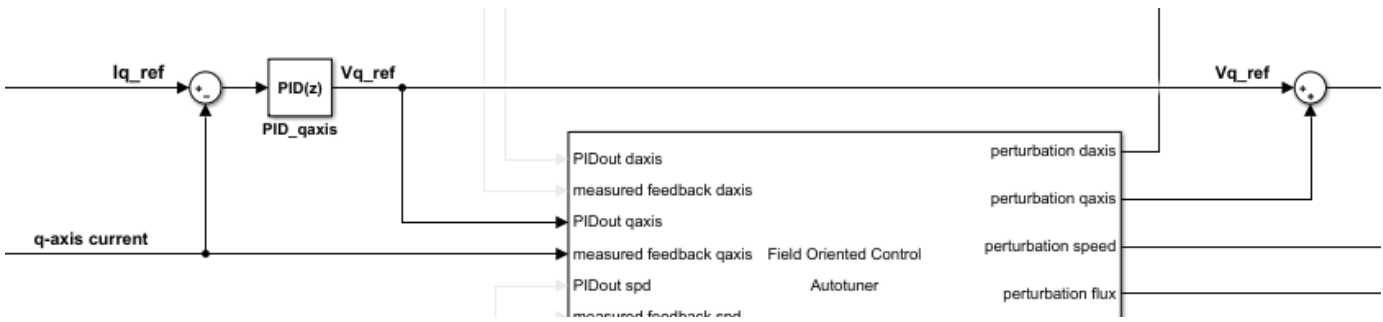
To enable this port, select **Tune D-axis current loop**.

Data Types: `single` | `double`

#### **PIDout\_qaxis** — Signal from quadrature-axis current controller

scalar

This port accepts the output of the q-axis controller `PID_qaxis`, which is the output of PID controller that regulates the q-axis current of the motor. The controller generates the q-axis voltage reference `Vq_ref`, while the FOC autotuner block generates perturbations used during the tuning experiment for the q-axis current loop.



**Dependencies**

To enable this port, select **Tune Q-axis current loop**.

Data Types: single | double

**measured feedback\_qaxis – Measured quadrature-axis current**

scalar

This port accepts the q-axis current obtained from the measured (sensed or estimated) motor currents.

**Dependencies**

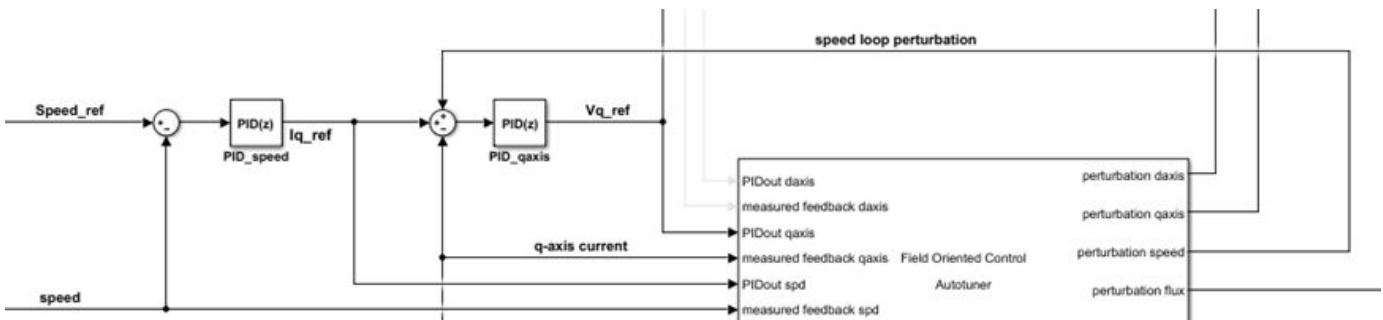
To enable this port, select **Tune Q-axis current loop**.

Data Types: single | double

**PIDout\_spd – Signal from speed controller**

scalar

This port accepts the output of the speed controller PID\_speed, which is the output of PID controller that regulates the speed of the motor. The controller generates the q-axis current reference Iq\_ref, while the FOC autotuner block generates perturbations used during the tuning experiment for the speed loop.



**Dependencies**

To enable this port, select **Tune speed loop**.

Data Types: single | double

**measured feedback\_spd – Measured speed**

scalar



This port accepts the measured (sensed or estimated) speed from the motor.

#### Dependencies

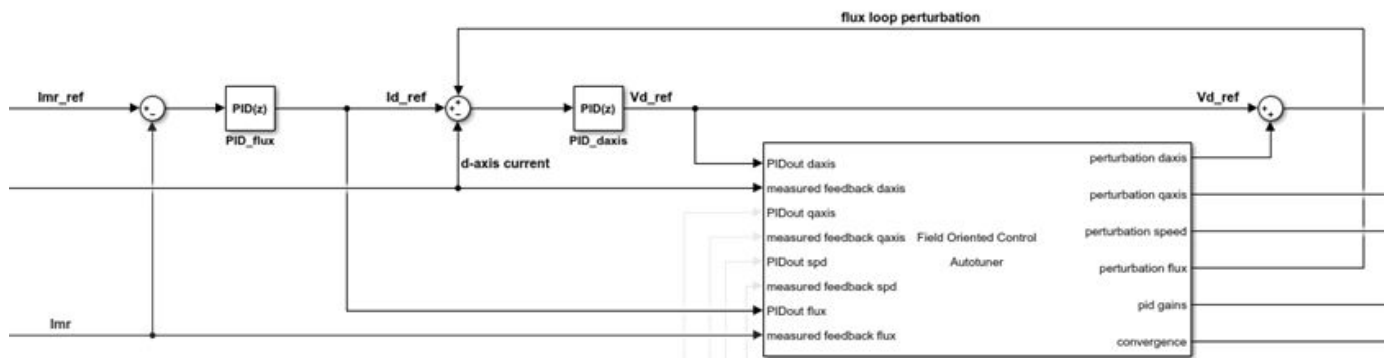
To enable this port, select **Tune speed loop**.

Data Types: `single` | `double`

#### PIDout\_flux – Signal from flux controller

scalar

This port accepts the output of the flux controller `PID_flux`, which is the output of PID controller that regulates the flux of the motor. The controller generates the d-axis current reference `Id_ref`, while the FOC autotuner block generates perturbations used during the tuning experiment for the flux loop.



For a permanent magnet synchronous motor (PMSM), there is no flux loop controller as the rotor flux is fixed and `Id_ref` is set to zero. In some applications you can provide a negative `Id_ref` value to implement field-weakening control and achieve higher rotor speeds at the cost of a higher current.

#### Dependencies

To enable this port, select **Tune flux loop**.

Data Types: `single` | `double`

#### measured\_feedback\_flux – Measured flux

scalar

This port accepts the measured (sensed or estimated) flux from the motor.

#### Dependencies

To enable this port, select **Tune flux loop**.

Data Types: `single` | `double`

#### start/stop – Start and stop autotuning experiment

scalar

To externally start and stop the autotuning process, provide a signal at the `start/stop` port and the `ActiveLoop` port.

- The experiment starts when the value of the signal changes from negative or zero to positive.

- The experiment stops when the value of the signal changes from positive to negative or zero.

For the duration of the experiment, for each loop, the block injects sinusoidal perturbations at the plant input associated with the loop, near the nominal operating point, to collect input-output data and estimate frequency response. When the experiment stops, the block computes PID gains based on the plant frequency responses estimated near the target bandwidth.

When the experiment is not running, the block does not inject any perturbations at the plant inputs. In this state, the block has no impact on plant or controller behavior.

Typically, you can use a signal that changes from 0 to 1 to start the experiment, and from 1 to 0 to stop it. Consider the following when you configure the `start/stop` signal.

- Start the experiment when the motor is at the desired equilibrium operating point. Use the initial controller to drive the motor to the operating point.
- Avoid any input or output disturbance on the motor during the experiment. If your existing closed-loop system has good disturbance rejection, then the experiment can handle small disturbances. Otherwise, large disturbances can distort the plant output and reduce the accuracy of the frequency-response estimation.
- Let the experiment run long enough for the algorithm to collect sufficient data for a good estimate at all frequencies it probes. There are two ways to determine when to stop the experiment:
  - Determine the experiment duration in advance. A conservative estimate for the experiment duration is  $200/\omega_c$  in superposition experiment mode or  $550/\omega_c$  in sinestream experiment mode, where  $\omega_c$  is your target bandwidth.
  - Observe the signal at the convergence output, and stop the experiment when the signal stabilizes near 100%.
- When you stop the experiment, the block computes tuned PID gains and updates the signal at the `pid gains` port.

You can configure any logic appropriate for your application to control the start and stop times of the experiment. The `start/stop` signal is specified along with `ActiveLoop`. `ActiveLoop` takes integer values 1 to 4 and specifies which loop to tune.

Alternatively, if you are tuning in simulation or external mode, you can specify the tuning experiment sequence, start time and duration in the block parameters.

### Dependencies

To enable this port, on the Block tab under **Parameters Source**, select **Use external source for start/stop of experiment**.

Data Types: `single` | `double`

### ActiveLoop — Specify active loop for autotuning experiment

scalar

Set the `ActiveLoop` value to specify which loop to tune when providing an external source for the start and stop times of the tuning experiment.

ActiveLoop Value	Loop to Tune
1	D-axis current loop

ActiveLoop Value	Loop to Tune
2	Q-axis current loop
3	Speed loop
4	Flux loop

You can configure any logic appropriate for your application along with the `start/stop` port to control the sequence and the time at which the loop tuning experiment runs. `ActiveLoop` takes integer values from 1 to 4 and specifies which loop to tune. Any other number will result in no tuning taking place regardless of the `start/stop` signal. For example, when you supply a constant value 2 at `ActiveLoop` and the signal at `start/stop` rises, the block starts the tuning experiment for the q-axis current loop.

Alternatively, you can specify the tuning experiment sequence, start time, and duration in the block parameters.

#### Dependencies

To enable this port, on the Block tab under **Parameters Source**, select **Use external source for start/stop of experiment**.

Data Types: `single` | `double`

#### bandwidth — Target bandwidth for tuning

scalar | vector | bus

Supply the values for the `Target bandwidth (rad/sec)` parameter for each loop to be tuned. If you are tuning multiple loops, you can specify the bandwidth as a vector or bus, entries of which correspond to the target bandwidth for the loops in this order:

- D-axis current loop
- Q-axis current loop
- Speed loop
- Flux loop

The vector signal must be specified as a N-by-1 or 1-by-N signal or if specified as a bus must have N elements, where N is the number of loops to be tuned. For instance, if you are tuning the q-axis current loop and the speed loop, and you specify a vector [5000, 200] at this port, the block tunes the q-axis current controller with the target bandwidth 5000 rad/sec and the speed loop controller with the target bandwidth 200 rad/sec.

If you are tuning multiple loops and specify a scalar value at this port, then the block uses the same target bandwidth to tune all the controllers. For effective cascade control, the inner control loops (d-axis and q-axis) must respond much faster than the outer control loops (flux and speed). Therefore, you must supply the target bandwidth as a vector or bus signal when tuning multiple loops.

Alternatively, you can specify target bandwidth for individual loops in block parameters. For more information on how to choose a bandwidth, see that parameter description.

#### Dependencies

To enable this port, on the Block tab under **Parameters Source**, select **Use external source for bandwidth**.

Data Types: `single` | `double`

**target PM — Target phase margin for tuning**

scalar | vector | bus

Supply a value for the **Target phase margin (degrees)** parameter for each loop to be tuned. If you are tuning multiple loops, you can specify **target PM** as a vector or bus, entries of which correspond to the target phase margin for the loops in this order:

- D-axis current loop
- Q-axis current loop
- Speed loop
- Flux loop

The vector signal must be specified as a N-by-1 or 1-by-N signal or if specified as a bus must have N elements, where N is the number of loops to be tuned. For instance, if you are tuning q-axis current loop and speed loop, and you specify a vector [60, 45] at this port, the block tunes q-axis current controller with target phase margin 60 degrees and speed loop controller with target phase margin 45 degrees.

If you are tuning multiple loops and specify a scalar value at this port, then the block uses the same target phase margin to tune all the controllers.

Alternatively, you can specify target phase margin for individual loops in block parameters. For more information on how to choose a target phase margin, see that parameter description.

**Dependencies**

To enable this port, on the Block tab under **Parameters Source**, select **Use external source for target phase margin**.

Data Types: single | double

**sine Amp — Amplitudes of injected sinusoidal perturbation signals**

vector | matrix

Supply a value for the **Sine Amplitudes** parameter for each loop to be tuned. Specify one of the following:

- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning.
- N-by-5 matrix, where N is the number of loops to be tuned. Each row entry must be of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$ .

If you are tuning multiple loops and specify a vector of length 5 at this port, then the block uses the specified amplitude for all the loops at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$  corresponding to that loop.

Alternatively, you can specify the sinusoidal perturbation amplitude for individual loops in block parameters. For more information, see the parameter description.

**Dependencies**

To enable this port, on the Block tab under **Parameters Source**, select **Use external source for sine amplitudes**.

Data Types: single | double

## Output

### **perturbation\_daxis — Direct-axis current input perturbation**

scalar

Perturbation signal input used for estimating the frequency-response data model associated with the d-axis current control loop. Inject the perturbation signal from this port by using a sum block to the output of the PID controller that regulates the d-axis current.

- When the experiment is running, the block generates perturbation signals at this port.
- When the experiment is not running, the signal at this port is zero. In this state, the block has no effect on the plant.

### **Dependencies**

To enable this port, select **Tune D-axis current loop**.

Data Types: `single` | `double`

### **perturbation\_qaxis — Quadrature-axis current input perturbation**

scalar

Perturbation signal input used for estimating the frequency-response data model associated with the q-axis current control loop. Inject this perturbation signal from this port by using a sum block to the output of the PID controller that regulates the q-axis current.

- When the experiment is running, the block generates perturbation signals at this port.
- When the experiment is not running, the signal at this port is zero. In this state, the block has no effect on the plant.

### **Dependencies**

To enable this port, select **Tune Q-axis current loop**.

Data Types: `single` | `double`

### **perturbation\_spd — Speed input perturbation**

scalar

Perturbation signal input used for estimating the frequency-response data model associated with the motor speed control loop. Inject this perturbation signal from this port by using a sum block with the output of the PID controller that regulates the speed of the motor.

- When the experiment is running, the block generates perturbation signals at this port.
- When the experiment is not running, the signal at this port is zero. In this state, the block has no effect on the plant.

### **Dependencies**

To enable this port, select **Tune speed loop**.

Data Types: `single` | `double`

### **perturbation\_flux — Flux input perturbation**

scalar

Perturbation signal input used for estimating the frequency-response data model associated with the motor flux control loop. Inject this perturbation signal from this port by using a sum block to the output of the PID controller that regulates the flux linkage of the motor.

- When the experiment is running, the block generates perturbation signals at this port.
- When the experiment is not running, the signal at this port is zero. In this state, the block has no effect on the plant.

### Dependencies

To enable this port, select **Tune flux loop**.

Data Types: `single` | `double`

### pid gains — Tuned PID coefficients

bus

This 4-element bus signal contains the tuned PID gains  $P$ ,  $I$ ,  $D$ , and the filter coefficient  $N$  for each control loop the block tunes. These values correspond to the  $P$ ,  $I$ ,  $D$ , and  $N$  parameters in the expressions given in the **Form** parameter. Initially, the values are 0, 0, 0, and 100, respectively. The block updates the values when the experiment ends. The bus signal corresponding to each loop the block tunes always has four elements, even if you are not tuning a PIDF controller.

Data Types: `single` | `double`

### convergence — Convergence of FRD estimation during experiment

scalar

The block uses perturbation signals to estimate the frequency response of the plant associated with each loop at several frequencies around the target bandwidth for tuning. **convergence** indicates how close to completion the estimation of the plant frequency response is. Typically, this value quickly rises to about 90% after the experiment begins, and then gradually converges to a higher value. Stop the experiment when it levels off near 100%.

Data Types: `single` | `double`

### estimated PM — Estimated phase margin for most recently tuned loop

scalar

This port outputs the estimated phase margin achieved by the tuned controller for the most recently tuned loop, in degrees. The block updates this value when the tuning experiment ends for each loop. The estimated phase margin is calculated from the angle of  $G(j\omega_c)C(j\omega_c)$ , where  $G$  is the estimated plant for that loop,  $C$  is the tuned controller, and  $\omega_c$  is the crossover frequency (bandwidth). The estimated phase margin might differ from the target phase margin specified by the **Target phase margin (degrees)** parameter. It is an indicator of the robustness and stability achieved by the tuned system.

- Typically, the estimated phase margin is near the target phase margin. In general, the larger the value, the more robust is the tuned system, and the less overshoot there is.
- A negative phase margin indicates that the closed-loop system might be unstable.

### Dependencies

To enable this port, on the Block tab, select **Estimated phase margin achieved by tuned controllers**.

Data Types: `single` | `double`

### **frd** — Estimated frequency response for most recently tuned loop

vector

This port outputs the frequency-response data estimated by the experiment for most recently tuned loop. Initially, the value at `frd` is `[0, 0, 0, 0, 0]`. During the experiment, the block injects signals at frequencies `[1/10, 1/3, 1, 3, 10] $\omega_c$` , where  $\omega_c$  is the target bandwidth. At each sample time during the experiment, the block updates `frd` with a vector containing the complex frequency response at each of these frequencies. You can use the progress of the response as an alternative to `convergence` to examine the convergence of the estimation. When the experiment stops, the block updates `frd` with the final estimated frequency response used for computing the PID gains.

#### **Dependencies**

To enable this port, on the Block tab, select **Plant frequency responses near bandwidth**.

Data Types: `single` | `double`

### **nominal** — Plant input and output at nominal operating point for most recently tuned loop

vector

This port outputs a vector containing the plant input and plant output for the most recently tuned loop or the loop currently being tuned. These values are the plant input and output at the nominal operating point at which the block performs the experiment.

#### **Dependencies**

To enable this port, on the Block tab, select **Plant nominal input and output**.

Data Types: `single` | `double`

### **loop\_startstops** — Active loop

bus

This 4-element bus signal indicates whether the tuning experiment for each loop tuned by the block is active or not. For each signal in the bus, the port outputs the logical value `1` (`true`) for the loop when the tuning experiment is running. The value is logical `0` (`false`) when the experiment is over or has not yet started. You can use this port to trigger updates of PID gains for individual loops.

#### **Dependencies**

To enable this port, on the Block tab, disable **Use external source for start/stop of experiment** and select **Start/stop of autotuning process**.

Data Types: `single` | `double`

## **Parameters**

### **Tune D-axis current loop** — Enable d-axis current loop tuning

`on` (default) | `off`

Use this parameter to enable or disable d-axis current loop autotuning.

#### **Programmatic Use**

**Block Parameter:** `TuneDaxisLoop`

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### **Tune Q-axis current loop — Enable q-axis current loop tuning**

on (default) | off

Use this parameter to enable or disable q-axis current loop autotuning.

#### **Programmatic Use**

**Block Parameter:** TuneQaxisLoop

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### **Tune speed loop — Enable speed loop tuning**

on (default) | off

Use this parameter to enable or disable speed loop autotuning.

#### **Programmatic Use**

**Block Parameter:** TuneSpeedLoop

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### **Tune flux loop — Enable flux loop tuning**

on (default) | off

Use this parameter to enable or disable flux loop autotuning.

#### **Programmatic Use**

**Block Parameter:** TuneSpeedLoop

**Type:** character vector

**Values:** 'on' | 'off'

**Default:** 'on'

### **Use same settings for current loop controllers (D-axis + Q-axis) — Enable same tuning and experiment settings for direct-axis and quadrature-axis current loops**

off (default) | on

Select this parameter to enable the same tuning and experiment settings for d-axis and q-axis current loops. When enabled, the block uses the same controller settings, target bandwidth, phase margin, and other experiment settings to tune d-axis and q-axis current loops.

#### **Programmatic Use**

**Block Parameter:** UseSameSettingsInner

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### **Use same settings for outer loop controllers (Speed + Flux) — Enable same tuning and experiment settings for speed and flux loops**

off (default) | on



Select this parameter to enable the same tuning and experiment settings for speed and flux loops. When enabled, the block uses the same controller settings, target bandwidth, phase margin, and other experiment settings to tune speed and flux loops.

**Programmatic Use**

**Block Parameter:** UseSameSettingsOuter

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Experiment sample time — Sample time of frequency response estimation experiment**

-1 (default) | positive scalar

Specify the sample time of the frequency response estimation experiment performed by the block in seconds.

By default, the experiment sample time is set to inherited (-1) and the block performs the frequency response estimation experiment, for each loop, at the inherited sample time. Use this parameter to specify a sample time for the frequency response estimation experiment that is different from the tuning and PID gain calculation sample rates. For each loop that you tune, the frequency responses are estimated at the sample time specified in this parameter.

When you specify different sample times for tuning, experiment, and loops, you can configure Simulink to treat each block module rate as a separate task to enable multitasking execution for your model. This multitasking mode helps improve performance on hardware. For more information, see “Treat each discrete rate as a separate task”.

**Programmatic Use**

**Block Parameter:** TsExperiment

**Type:** scalar

**Value:** positive scalar

**Default:** -1 (inherited)

**Tuning Tab**

**Use different sample time for tuning — Enable tuning at different sample time from loop PID controller and experiment**

off (default) | on

By default, the block runs tuning for each loop at the same sample time that you specify in the **Controller sample time (sec)** parameter for that loop. Enable this parameter to run tuning at a sample rate that is different from the sample rate of the PID controllers you are tuning and the frequency response estimation experiment performed by the block. The PID gain tuning algorithm is computationally intensive, and when you want to deploy the block to hardware and tune a controller with a fast sample time, some hardware might not complete the PID gain calculation in a single time step. To reduce the hardware throughput requirements, specify a tuning sample time slower than the controller sample time using the **Tuning sample time (sec)** parameter.

**Programmatic Use**

**Block Parameter:** UseTuningTs

**Type:** character vector

**Value:** 'off' | 'on'

**Default:** 'off'

**Tuning sample time (sec) — Sample time of tuning algorithm**

0.2 (default) | positive scalar

Specify the sample time of the tuning algorithm in seconds.

If you intend to deploy the block on hardware with limited processing power and want to tune a controller with a fast sample time, specify a sample time such that the tuning algorithm runs at a slower rate than the PID controllers you are tuning. For each loop that you tune, after the frequency response estimation experiment ends, controller tuning occurs at the sample time specified in this parameter.

#### Dependencies

To enable this parameter, select **Use different sample time for tuning**.

#### Programmatic Use

**Block Parameter:** TsTuning

**Type:** scalar

**Value:** positive scalar

**Default:** 0.2

#### D-axis Current Loop

##### Type — D-axis current loop PID controller actions

PI (default) | PID | PIDF | ...

Specify the type of PID controller associated with the d-axis current control loop.

The controller type indicates what actions are present in the controller that regulates the loop. The following controller types are available for PID autotuning:

- P — Proportional only
- I — Integral only
- PI — Proportional and integral
- PD — Proportional and derivative
- PDF — Proportional and derivative with derivative filter
- PID — Proportional, integral, and derivative
- PIDF — Proportional, integral, and derivative with derivative filter

Make sure the controller type matches the controller that regulates the loop.

#### Programmatic Use

**Block Parameter:** PIDTypeDaxis

**Type:** character vector

**Values:** 'P' | 'I' | 'PI' | 'PD' | 'PDF' | 'PID' | 'PIDF'

**Default:** 'PI'

##### Form — D-axis current loop PID controller form

Parallel (default) | Ideal

Specify the PID controller form associated with your d-axis current control loop.

The controller form determines the interpretation of the PID coefficients  $P$ ,  $I$ ,  $D$ , and  $N$ .

- **Parallel** — In **Parallel** form, the transfer function of a discrete-time PIDF controller is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right]$$

where  $F_i(z)$  and  $F_d(z)$  are the integrator and filter formulas (see `Integrator` method and `Filter` method).

Other controller actions amount to setting  $P$ ,  $I$ , or  $D$  to zero.

- `Ideal` — In `Ideal` form, the transfer function of a discrete-time PIDF controller is

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

Other controller actions amount to setting  $D$  to zero or setting  $I$  to `Inf`. (In ideal form, the controller must have proportional action.)

Make sure the controller form matches the controller that regulates the loop.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** `PIDFormDaxis`

**Type:** character vector

**Values:** 'Parallel' | 'Ideal'

**Default:** 'Parallel'

**Controller sample time (sec) — D-axis current loop PID controller sample time**

0.001 (default) | positive scalar | -1

Specify the sample time of your PID controller associated with the d-axis current control loop in seconds. This parameter sets the sample time used to calculate the PID controller gains for the loop.

To perform PID tuning, the block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$   $\omega_c$  is the controller sample time that you specify with the `Controller sample time (sec)` parameter.

Make sure the controller sample time matches the controller that regulates the loop.

**Tips**

If you want to run the deployed block with different sample times in your application, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the desired sample time. If you do not plan to change the sample time after deployment, specify a fixed and finite sample time.

**Programmatic Use**

**Block Parameter:** `TsDaxis`

**Type:** scalar

**Value** positive scalar | -1

**Default:** 0.001

**Integrator method — D-axis current loop controller discrete integration formula for integrator term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the integrator term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Integrator method parameter determines the formula  $F_i$  as follows.

Integrator method	$F_i$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller integrator method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes integral action.

**Programmatic Use**

**Block Parameter:** IntegratorMethodDaxis

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Filter method — D-axis current loop controller discrete integration formula for derivative filter term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the derivative filter term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Filter method parameter determines the formula  $F_d$  as follows.

Filter method	$F_d$
Forward Euler	$\frac{T_s}{z-1}$
Backward Euler	$\frac{T_s z}{z-1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller derivative filter method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes derivative action with a derivative filter term.

**Programmatic Use**

**Block Parameter:** FilterMethodDaxis

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Target bandwidth (rad/sec) – D-axis current loop target crossover frequency of tuned response**

100 (default) | positive scalar

The target bandwidth is the target value for the 0 dB gain crossover frequency of the tuned open-loop response  $CP$ , where  $P$  is the plant response associated with the loop, and  $C$  is the controller response. This crossover frequency roughly sets the control bandwidth. For a rise time  $\tau$  seconds, a good guess for the target bandwidth is  $2/\tau$  rad/sec.

To perform PID tuning, the autotuner block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter. Because of this condition, the fastest rise time you can enforce for tuning is about  $6.67T_s$ . If this rise time does not meet your design goals, consider reducing  $T_s$ .

For best results, use a target bandwidth that is within about a factor of 10 of the bandwidth with the initial PID controller. To tune a controller for a larger change in bandwidth, tune incrementally using smaller changes.

To provide the target bandwidth by using an input port, on the Block tab, select **Use external source for bandwidth**.

**Programmatic Use**

**Block Parameter:** BandwidthDaxis

**Type:** positive scalar

**Default:** 100

**Target phase margin (degrees) – D-axis current loop target minimum phase margin**

60 (default) | scalar in range 0-90

Specify a target minimum phase margin for the tuned open-loop response associated with the d-axis current control loop at the crossover frequency.

The target phase margin reflects the desired robustness of the tuned system. Typically, choose a value in the range of about 45°-60°. In general, a higher phase margin reduces overshoot, but can limit the response speed. The default value 60° tends to balance performance and robustness, yielding about 5-10% overshoot, depending on the characteristics of your plant.

To provide the target phase margin by using an input port, on the Block tab, select **Use external source for target phase margins**.

**Tunable:** Yes**Programmatic Use****Block Parameter:** TargetPMDaxis**Type:** scalar**Values:** 0-90**Default:** 60**Q-axis Current Loop****Type – Q-axis current loop PID controller actions**

PI (default) | PID | PIDF | ...

Specify the type of PID controller associated with the q-axis current control loop.

The controller type indicates what actions are present in the controller that regulates the loop. The following controller types are available for PID autotuning:

- P – Proportional only
- I – Integral only
- PI – Proportional and integral
- PD – Proportional and derivative
- PDF – Proportional and derivative with derivative filter
- PID – Proportional, integral, and derivative
- PIDF – Proportional, integral, and derivative with derivative filter

Make sure the controller type matches the controller that regulates the loop.

**Programmatic Use****Block Parameter:** PIDTypeQaxis**Type:** character vector**Values:** 'P' | 'I' | 'PI' | 'PD' | 'PDF' | 'PID' | 'PIDF'**Default:** 'PI'**Form – Q-axis current loop PID controller form**

Parallel (default) | Ideal

Specify the PID controller form associated with your q-axis current control loop.

The controller form determines the interpretation of the PID coefficients  $P$ ,  $I$ ,  $D$ , and  $N$ .

- **Parallel** — In **Parallel** form, the transfer function of a discrete-time PIDF controller is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

where  $F_i(z)$  and  $F_d(z)$  are the integrator and filter formulas (see **Integrator method** and **Filter method**).

Other controller actions amount to setting  $P$ ,  $I$ , or  $D$  to zero.

- **Ideal** — In **Ideal** form, the transfer function of a discrete-time PIDF controller is

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

Other controller actions amount to setting  $D$  to zero or setting  $I$  to **Inf**. (In ideal form, the controller must have proportional action.)

Make sure the controller form matches the controller that regulates the loop.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** PIDFormQaxis

**Type:** character vector

**Values:** 'Parallel' | 'Ideal'

**Default:** 'Parallel'

**Controller sample time (sec) — Q-axis current loop PID controller sample time**

0.001 (default) | positive scalar | -1

Specify the sample time of your PID controller associated with the q-axis current control loop in seconds. This parameter sets the sample time used to calculate the PID controller gains for the loop.

To perform PID tuning, the block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$   $\omega_c$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter.

Make sure the controller sample time matches the controller that regulates the loop.

**Tips**

If you want to run the deployed block with different sample times in your application, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the desired sample time. If you do not plan to change the sample time after deployment, specify a fixed and finite sample time.

**Programmatic Use**

**Block Parameter:** TsQaxis

**Type:** scalar

**Value** positive scalar | -1

**Default:** 0.001

**Integrator method — Q-axis current loop controller discrete integration formula for integrator term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the integrator term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D\left[\frac{N}{1 + NF_d(z)}\right],$$

in parallel form, or in ideal form,

$$C = P\left[1 + IF_i(z) + D\left(\frac{N}{1 + NF_d(z)}\right)\right].$$

For a controller sample time  $T_s$ , the Integrator method parameter determines the formula  $F_i$  as follows.

Integrator method	$F_i$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller integrator method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes integral action.

**Programmatic Use**

**Block Parameter:** IntegratorMethodQaxis

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Filter method — Q-axis current loop controller discrete integration formula for derivative filter term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the derivative filter term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D\left[\frac{N}{1 + NF_d(z)}\right],$$

in parallel form, or in ideal form,

$$C = P\left[1 + IF_i(z) + D\left(\frac{N}{1 + NF_d(z)}\right)\right].$$



For a controller sample time  $T_s$ , the **Filter method** parameter determines the formula  $F_d$  as follows.

Filter method	$F_d$
Forward Euler	$\frac{T_s}{z-1}$
Backward Euler	$\frac{T_s z}{z-1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller derivative filter method matches the controller that regulates the loop.

**Tunable:** Yes

#### Dependencies

This parameter is enabled when the controller includes derivative action with a derivative filter term.

#### Programmatic Use

**Block Parameter:** FilterMethodQaxis

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

#### Target bandwidth (rad/sec) — Q-axis current loop target crossover frequency of tuned response

100 (default) | positive scalar

The target bandwidth is the target value for the 0 dB gain crossover frequency of the tuned open-loop response  $CP$ , where  $P$  is the plant response associated with the loop, and  $C$  is the controller response. This crossover frequency roughly sets the control bandwidth. For a rise time  $\tau$  seconds, a good guess for the target bandwidth is  $2/\tau$  rad/sec.

To perform PID tuning, the autotuner block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter. Because of this condition, the fastest rise time you can enforce for tuning is about  $6.67T_s$ . If this rise time does not meet your design goals, consider reducing  $T_s$ .

For best results, use a target bandwidth that is within about a factor of 10 of the bandwidth with the initial PID controller. To tune a controller for a larger change in bandwidth, tune incrementally using smaller changes.

To provide the target bandwidth by using an input port, on the Block tab, select **Use external source for bandwidth**.

#### Programmatic Use

**Block Parameter:** BandwidthQaxis

**Type:** positive scalar

**Default:** 100

### **Target phase margin (degrees) — Q-axis current loop target minimum phase margin**

60 (default) | scalar in range 0-90

Specify a target minimum phase margin for the tuned open-loop response associated with the q-axis current control loop at the crossover frequency.

The target phase margin reflects the desired robustness of the tuned system. Typically, choose a value in the range of about 45°-60°. In general, a higher phase margin reduces overshoot, but can limit the response speed. The default value 60° tends to balance performance and robustness, yielding about 5-10% overshoot, depending on the characteristics of your plant.

To provide the target phase margin by using an input port, on the Block tab, select **Use external source for target phase margins**.

**Tunable:** Yes

#### **Programmatic Use**

**Block Parameter:** TargetPMQaxis

**Type:** scalar

**Values:** 0-90

**Default:** 60

#### **Speed Loop**

### **Type — Speed loop PID controller actions**

PI (default) | PID | PIDF | ...

Specify the type of PID controller associated with the speed control loop.

The controller type indicates what actions are present in the controller that regulates the loop. The following controller types are available for PID autotuning:

- P — Proportional only
- I — Integral only
- PI — Proportional and integral
- PD — Proportional and derivative
- PDF — Proportional and derivative with derivative filter
- PID — Proportional, integral, and derivative
- PIDF — Proportional, integral, and derivative with derivative filter

Make sure the controller type matches the controller that regulates the loop.

#### **Programmatic Use**

**Block Parameter:** PIDTypeSpeed

**Type:** character vector

**Values:** 'P' | 'I' | 'PI' | 'PD' | 'PDF' | 'PID' | 'PIDF'

**Default:** 'PI'

### **Form — Speed loop PID controller form**

Parallel (default) | Ideal

Specify the PID controller form associated with your speed control loop.

The controller form determines the interpretation of the PID coefficients  $P$ ,  $I$ ,  $D$ , and  $N$ .

- **Parallel** — In **Parallel** form, the transfer function of a discrete-time PIDF controller is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

where  $F_i(z)$  and  $F_d(z)$  are the integrator and filter formulas (see **Integrator method** and **Filter method**).

Other controller actions amount to setting  $P$ ,  $I$ , or  $D$  to zero.

- **Ideal** — In **Ideal** form, the transfer function of a discrete-time PIDF controller is

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

Other controller actions amount to setting  $D$  to zero or setting  $I$  to Inf. (In ideal form, the controller must have proportional action.)

Make sure the controller form matches the controller that regulates the loop.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** PIDFormSpeed

**Type:** character vector

**Values:** 'Parallel' | 'Ideal'

**Default:** 'Parallel'

**Controller sample time (sec) — Speed loop PID controller sample time**

0.1 (default) | positive scalar | -1

Specify the sample time of your PID controller associated with the speed control loop in seconds. This parameter sets the sample time used to calculate the PID controller gains for the loop.

To perform PID tuning, the block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$   $\omega_c$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter.

Make sure the controller sample time matches the controller that regulates the loop.

### Tips

If you want to run the deployed block with different sample times in your application, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the desired sample time. If you do not plan to change the sample time after deployment, specify a fixed and finite sample time.

**Programmatic Use**

**Block Parameter:** TsSpeed

**Type:** scalar

**Value** positive scalar | -1

**Default:** 0.1

### Integrator method — Speed loop controller discrete integration formula for integrator term

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the integrator term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Integrator method parameter determines the formula  $F_i$  as follows.

Integrator method	$F_i$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller integrator method matches the controller that regulates the loop.

**Tunable:** Yes

#### Dependencies

This parameter is enabled when the controller includes integral action.

#### Programmatic Use

**Block Parameter:** IntegratorMethodSpeed

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

### Filter method — Speed loop controller discrete integration formula for derivative filter term

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the derivative filter term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the **Filter method** parameter determines the formula  $F_d$  as follows.

<b>Filter method</b>	$F_d$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller derivative filter method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes derivative action with a derivative filter term.

**Programmatic Use**

**Block Parameter:** FilterMethodSpeed

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Target bandwidth (rad/sec) — Speed loop target crossover frequency of tuned response**  
1 (default) | positive scalar

The target bandwidth is the target value for the 0 dB gain crossover frequency of the tuned open-loop response  $CP$ , where  $P$  is the plant response associated with the loop, and  $C$  is the controller response. This crossover frequency roughly sets the control bandwidth. For a rise time  $\tau$  seconds, a good guess for the target bandwidth is  $2/\tau$  rad/sec.

To perform PID tuning, the autotuner block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter. Because of this condition, the fastest rise time you can enforce for tuning is about  $6.67T_s$ . If this rise time does not meet your design goals, consider reducing  $T_s$ .

For best results, use a target bandwidth that is within about a factor of 10 of the bandwidth with the initial PID controller. To tune a controller for a larger change in bandwidth, tune incrementally using smaller changes.

To provide the target bandwidth by using an input port, on the Block tab, select **Use external source for bandwidth**.

**Programmatic Use****Block Parameter:** BandwidthSpeed**Type:** positive scalar**Default:** 1**Target phase margin (degrees) — Speed loop target minimum phase margin**

60 (default) | scalar in range 0-90

Specify a target minimum phase margin for the tuned open-loop response associated with the speed control loop at the crossover frequency.

The target phase margin reflects the desired robustness of the tuned system. Typically, choose a value in the range of about 45°–60°. In general, a higher phase margin reduces overshoot, but can limit the response speed. The default value 60° tends to balance performance and robustness, yielding about 5–10% overshoot, depending on the characteristics of your plant.

To provide the target phase margin by using an input port, on the Block tab, select **Use external source for target phase margins**.

**Tunable:** Yes**Programmatic Use****Block Parameter:** TargetPMSpeed**Type:** scalar**Values:** 0–90**Default:** 60**Flux Loop****Type — Flux loop PID controller actions**

PI (default) | PID | PIDF | ...

Specify the type of PID controller associated with the flux control loop.

The controller type indicates what actions are present in the controller that regulates the loop. The following controller types are available for PID autotuning:

- P — Proportional only
- I — Integral only
- PI — Proportional and integral
- PD — Proportional and derivative
- PDF — Proportional and derivative with derivative filter
- PID — Proportional, integral, and derivative
- PIDF — Proportional, integral, and derivative with derivative filter

Make sure the controller type matches the controller that regulates the loop.

**Programmatic Use****Block Parameter:** PIDTypeFlux**Type:** character vector**Values:** 'P' | 'I' | 'PI' | 'PD' | 'PDF' | 'PID' | 'PIDF'**Default:** 'PI'

**Form — Flux loop PID controller form**

Parallel (default) | Ideal

Specify the PID controller form associated with your flux control loop.

The controller form determines the interpretation of the PID coefficients  $P$ ,  $I$ ,  $D$ , and  $N$ .

- **Parallel** — In **Parallel** form, the transfer function of a discrete-time PIDF controller is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

where  $F_i(z)$  and  $F_d(z)$  are the integrator and filter formulas (see **Integrator method** and **Filter method**).

Other controller actions amount to setting  $P$ ,  $I$ , or  $D$  to zero.

- **Ideal** — In **Ideal** form, the transfer function of a discrete-time PIDF controller is

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

Other controller actions amount to setting  $D$  to zero or setting  $I$  to Inf. (In ideal form, the controller must have proportional action.)

Make sure the controller form matches the controller that regulates the loop.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** PIDFormFlux

**Type:** character vector

**Values:** 'Parallel' | 'Ideal'

**Default:** 'Parallel'

**Controller sample time (sec) — Flux loop PID controller sample time**

0.1 (default) | positive scalar | -1

Specify the sample time of your PID controller associated with the flux control loop in seconds. This parameter sets the sample time used to calculate the PID controller gains for the loop.

To perform PID tuning, the block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$   $\omega_c$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter.

Make sure the controller sample time matches the controller that regulates the loop.

**Tips**

If you want to run the deployed block with different sample times in your application, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the desired sample time. If you do not plan to change the sample time after deployment, specify a fixed and finite sample time.

**Programmatic Use**

**Block Parameter:** TsFlux

**Type:** scalar

**Value:** positive scalar | -1

**Default:** 0.1

**Integrator method — Flux loop controller discrete integration formula for integrator term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the integrator term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Integrator method parameter determines the formula  $F_i$  as follows.

Integrator method	$F_i$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller integrator method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes integral action.

**Programmatic Use**

**Block Parameter:** IntegratorMethodFlux

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Filter method — Flux loop controller discrete integration formula for derivative filter term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the derivative filter term in your controller. In discrete time, the PID controller transfer function assumed by the block is



$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Filter method parameter determines the formula  $F_d$  as follows.

Filter method	$F_d$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller derivative filter method matches the controller that regulates the loop.

**Tunable:** Yes

#### Dependencies

This parameter is enabled when the controller includes derivative action with a derivative filter term.

#### Programmatic Use

**Block Parameter:** FilterMethodFlux

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

#### Target bandwidth (rad/sec) — Flux loop target crossover frequency of tuned response

1 (default) | positive scalar

The target bandwidth is the target value for the 0 dB gain crossover frequency of the tuned open-loop response  $CP$ , where  $P$  is the plant response associated with the loop, and  $C$  is the controller response. This crossover frequency roughly sets the control bandwidth. For a rise time  $\tau$  seconds, a good guess for the target bandwidth is  $2/\tau$  rad/sec.

To perform PID tuning, the autotuner block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter. Because of this condition, the fastest rise time you can enforce for tuning is about  $6.67T_s$ . If this rise time does not meet your design goals, consider reducing  $T_s$ .

For best results, use a target bandwidth that is within about a factor of 10 of the bandwidth with the initial PID controller. To tune a controller for a larger change in bandwidth, tune incrementally using smaller changes.

To provide the target bandwidth by using an input port, on the Block tab, select **Use external source for bandwidth**.

**Programmatic Use**

**Block Parameter:** BandwidthFlux

**Type:** positive scalar

**Default:** 1

**Target phase margin (degrees) — Flux loop target minimum phase margin**

60 (default) | scalar in range 0-90

Specify a target minimum phase margin for the tuned open-loop response associated with the flux control loop at the crossover frequency.

The target phase margin reflects the desired robustness of the tuned system. Typically, choose a value in the range of about 45°-60°. In general, a higher phase margin reduces overshoot, but can limit the response speed. The default value 60° tends to balance performance and robustness, yielding about 5-10% overshoot, depending on the characteristics of your plant.

To provide the target phase margin by using an input port, on the Block tab, select **Use external source for target phase margins**.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** TargetPMFlux

**Type:** scalar

**Values:** 0-90

**Default:** 60

**Current Loops (Q-axis + D-axis)**

**Type — Current loop PID controller actions**

PI (default) | PID | PIDF | ...

Specify the type of PID controller associated with the current control loops.

The controller type indicates what actions are present in the controller that regulates the loop. The following controller types are available for PID autotuning:

- P — Proportional only
- I — Integral only
- PI — Proportional and integral
- PD — Proportional and derivative
- PDF — Proportional and derivative with derivative filter
- PID — Proportional, integral, and derivative
- PIDF — Proportional, integral, and derivative with derivative filter

Make sure the controller type matches the controller that regulates the loop.

**Programmatic Use**

**Block Parameter:** PIDTypeAllInner

**Type:** character vector

**Values:** 'P' | 'I' | 'PI' | 'PD' | 'PDF' | 'PID' | 'PIDF'

**Default:** 'PI'

### Form — Current loop PID controller form

Parallel (default) | Ideal

Specify the PID controller form associated with your current control loops.

The controller form determines the interpretation of the PID coefficients  $P$ ,  $I$ ,  $D$ , and  $N$ .

- **Parallel** — In **Parallel** form, the transfer function of a discrete-time PIDF controller is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

where  $F_i(z)$  and  $F_d(z)$  are the integrator and filter formulas (see **Integrator method** and **Filter method**).

Other controller actions amount to setting  $P$ ,  $I$ , or  $D$  to zero.

- **Ideal** — In **Ideal** form, the transfer function of a discrete-time PIDF controller is

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

Other controller actions amount to setting  $D$  to zero or setting  $I$  to **Inf**. (In ideal form, the controller must have proportional action.)

Make sure the controller form matches the controller that regulates the loop.

**Tunable:** Yes

### Programmatic Use

**Block Parameter:** PIDFormAllInner

**Type:** character vector

**Values:** 'Parallel' | 'Ideal'

**Default:** 'Parallel'

### Controller sample time (sec) — Current loop PID controller sample time

0.001 (default) | positive scalar | -1

Specify the sample time of your PID controllers associated with the current control loops in seconds. This parameter sets the sample time used to calculate the PID controller gains for the loop.

To perform PID tuning, the block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$   $\omega_c$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter.

Make sure the controller sample time matches the controller that regulates the loop.

### Tips

If you want to run the deployed block with different sample times in your application, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the desired sample time. If you do not plan to change the sample time after deployment, specify a fixed and finite sample time.

**Programmatic Use**

**Block Parameter:** TsAllInner

**Type:** scalar

**Value:** positive scalar | -1

**Default:** 0.001

**Integrator method — Current loop controller discrete integration formula for integrator term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the integrator term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Integrator method parameter determines the formula  $F_i$  as follows.

Integrator method	$F_i$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller integrator method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes integral action.

**Programmatic Use**

**Block Parameter:** IntegratorMethodAllInner

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Filter method — Current loop controller discrete integration formula for derivative filter term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the derivative filter term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the **Filter method** parameter determines the formula  $F_d$  as follows.

Filter method	$F_d$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller derivative filter method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes derivative action with a derivative filter term.

**Programmatic Use**

**Block Parameter:** FilterMethodAllInner

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Target bandwidth (rad/sec) — Current loop target crossover frequency of tuned responses**

100 (default) | positive scalar

The target bandwidth is the target value for the 0 dB gain crossover frequency of the tuned open-loop response  $CP$ , where  $P$  is the plant response associated with the loop, and  $C$  is the controller response. This crossover frequency roughly sets the control bandwidth. For a rise time  $\tau$  seconds, a good guess for the target bandwidth is  $2/\tau$  rad/sec.

To perform PID tuning, the autotuner block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter. Because of this condition, the fastest rise time you can enforce for tuning is about  $6.67T_s$ . If this rise time does not meet your design goals, consider reducing  $T_s$ .

For best results, use a target bandwidth that is within about a factor of 10 of the bandwidth with the initial PID controller. To tune a controller for a larger change in bandwidth, tune incrementally using smaller changes.

To provide the target bandwidth by using an input port, on the Block tab, select **Use external source for bandwidth**.

**Programmatic Use**

**Block Parameter:** BandwidthAllInner

**Type:** positive scalar

**Default:** 1

**Target phase margin (degrees) — Current loop target minimum phase margins**

60 (default) | scalar in range 0-90

Specify target minimum phase margin for the tuned open-loop responses associated with the current control loops at the crossover frequency.

The target phase margin reflects the desired robustness of the tuned system. Typically, choose a value in the range of about 45°-60°. In general, a higher phase margin reduces overshoot, but can limit the response speed. The default value 60° tends to balance performance and robustness, yielding about 5-10% overshoot, depending on the characteristics of your plant.

To provide the target phase margin by using an input port, on the Block tab, select **Use external source for target phase margins**.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** TargetPMAAllInner

**Type:** scalar

**Values:** 0-90

**Default:** 60

**Outer Loops (Speed + Flux)**

**Type — Outer loop PID controller actions**

PI (default) | PID | PIDF | ...

Specify the type of PID controllers associated with the outer control loops.

The controller type indicates what actions are present in the controller that regulates the loop. The following controller types are available for PID autotuning:

- P — Proportional only
- I — Integral only
- PI — Proportional and integral
- PD — Proportional and derivative
- PDF — Proportional and derivative with derivative filter
- PID — Proportional, integral, and derivative
- PIDF — Proportional, integral, and derivative with derivative filter

Make sure the controller type matches the controller that regulates the loop.

**Programmatic Use****Block Parameter:** PIDTypeAllOuter**Type:** character vector**Values:** 'P' | 'I' | 'PI' | 'PD' | 'PDF' | 'PID' | 'PIDF'**Default:** 'PI'**Form — Outer loop PID controller form**

Parallel (default) | Ideal

Specify the PID controller form associated with your outer control loops.

The controller form determines the interpretation of the PID coefficients  $P$ ,  $I$ ,  $D$ , and  $N$ .

- **Parallel** — In **Parallel** form, the transfer function of a discrete-time PIDF controller is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

where  $F_i(z)$  and  $F_d(z)$  are the integrator and filter formulas (see **Integrator method** and **Filter method**).

Other controller actions amount to setting  $P$ ,  $I$ , or  $D$  to zero.

- **Ideal** — In **Ideal** form, the transfer function of a discrete-time PIDF controller is

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

Other controller actions amount to setting  $D$  to zero or setting  $I$  to  $\text{Inf}$ . (In ideal form, the controller must have proportional action.)

Make sure the controller form matches the controller that regulates the loop.

**Tunable:** Yes**Programmatic Use****Block Parameter:** PIDFormAllOuter**Type:** character vector**Values:** 'Parallel' | 'Ideal'**Default:** 'Parallel'**Controller sample time (sec) — Outer loop PID controller sample time**

0.1 (default) | positive scalar | -1

Specify the sample time of your PID controllers associated with the outer control loop in seconds. This parameter sets the sample time used to calculate the PID controller gains for the loop.

To perform PID tuning, the block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time that you specify with the **Controller sample time (sec)** parameter.

Make sure the controller sample time matches the controller that regulates the loop.

**Tips**

If you want to run the deployed block with different sample times in your application, set this parameter to -1 and put the block in a Triggered Subsystem. Then, trigger the subsystem at the

desired sample time. If you do not plan to change the sample time after deployment, specify a fixed and finite sample time.

**Programmatic Use**

**Block Parameter:** TsAllOuter

**Type:** scalar

**Value:** positive scalar | -1

**Default:** 0.1

**Integrator method — Outer loop controller discrete integration formula for integrator term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the integrator term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Integrator method parameter determines the formula  $F_i$  as follows.

Integrator method	$F_i$
Forward Euler	$\frac{T_s}{z - 1}$
Backward Euler	$\frac{T_s z}{z - 1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller integrator method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes integral action.

**Programmatic Use**

**Block Parameter:** IntegratorMethodAllOuter

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'



**Filter method – Outer loop controller discrete integration formula for derivative filter term**

Forward Euler (default) | Backward Euler | Trapezoidal

Specify the discrete integration formula for the derivative filter term in your controller. In discrete time, the PID controller transfer function assumed by the block is

$$C = P + IF_i(z) + D \left[ \frac{N}{1 + NF_d(z)} \right],$$

in parallel form, or in ideal form,

$$C = P \left[ 1 + IF_i(z) + D \left( \frac{N}{1 + NF_d(z)} \right) \right].$$

For a controller sample time  $T_s$ , the Filter method parameter determines the formula  $F_d$  as follows.

Filter method	$F_d$
Forward Euler	$\frac{T_s}{z-1}$
Backward Euler	$\frac{T_s z}{z-1}$
Trapezoidal	$\frac{T_s z + 1}{2 z - 1}$

For more information about the relative advantages of each method, see the Discrete PID Controller block reference page.

Make sure the controller derivative filter method matches the controller that regulates the loop.

**Tunable:** Yes

**Dependencies**

This parameter is enabled when the controller includes derivative action with a derivative filter term.

**Programmatic Use**

**Block Parameter:** FilterMethodAllOuter

**Type:** character vector

**Values:** 'Forward Euler' | 'Backward Euler' | 'Trapezoidal'

**Default:** 'Forward Euler'

**Target bandwidth (rad/sec) – Outer loop target crossover frequency of tuned responses**

1 (default) | positive scalar

The target bandwidth is the target value for the 0 dB gain crossover frequency of the tuned open-loop response  $CP$ , where  $P$  is the plant response associated with the loop, and  $C$  is the controller response. This crossover frequency roughly sets the control bandwidth. For a rise time  $\tau$  seconds, a good guess for the target bandwidth is  $2/\tau$  rad/sec.

To perform PID tuning, the autotuner block measures frequency-response information up to a frequency of 10 times the target bandwidth. To ensure that this frequency is less than the Nyquist frequency, the target bandwidth  $\omega_c$  must satisfy  $\omega_c T_s \leq 0.3$ , where  $T_s$  is the controller sample time

that you specify with the **Controller sample time (sec)** parameter. Because of this condition, the fastest rise time you can enforce for tuning is about  $6.67T_s$ . If this rise time does not meet your design goals, consider reducing  $T_s$ .

For best results, use a target bandwidth that is within about a factor of 10 of the bandwidth with the initial PID controller. To tune a controller for a larger change in bandwidth, tune incrementally using smaller changes.

To provide the target bandwidth by using an input port, on the Block tab, select **Use external source for bandwidth**.

**Programmatic Use**

**Block Parameter:** BandwidthAllOuter

**Type:** positive scalar

**Default:** 1

**Target phase margin (degrees) – Outer loop target minimum phase margins**

60 (default) | scalar in range 0-90

Specify a target minimum phase margin for the tuned open-loop responses associated with the outer control loops at the crossover frequency.

The target phase margin reflects the desired robustness of the tuned system. Typically, choose a value in the range of about  $45^\circ$ - $60^\circ$ . In general, a higher phase margin reduces overshoot, but can limit the response speed. The default value  $60^\circ$  tends to balance performance and robustness, yielding about 5-10% overshoot, depending on the characteristics of your plant.

To provide the target phase margin by using an input port, on the Block tab, select **Use external source for target phase margins**.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** TargetPMAllOuter

**Type:** scalar

**Values:** 0-90

**Default:** 60

**Experiment Tab**

**Experiment Start/Stop**

**D-axis current loop start time (sec) – Specify direct-axis current loop tuning experiment start time**

1 (default)

Specify the simulation time when the d-axis current loop tuning experiment starts.

**Programmatic Use**

**Block Parameter:** StartTimeDaxis

**Type:** positive scalar

**Default:** 1

**D-axis current loop experiment duration (sec) – Specify direct-axis current loop tuning experiment duration**

0.05 (default)

Specify the d-axis current loop tuning experiment duration.

**Programmatic Use**

**Block Parameter:** DurationDaxis

**Type:** positive scalar

**Default:** 0.05

**Q-axis current loop start time (sec) — Specify quadrature-axis current loop tuning experiment start time**

1.1 (default)

Specify the simulation time when the q-axis current loop tuning experiment starts.

**Programmatic Use**

**Block Parameter:** StartTimeQaxis

**Type:** positive scalar

**Default:** 1.1

**Q-axis current loop experiment duration (sec) — Specify quadrature-axis current loop tuning experiment duration**

0.05 (default)

Specify the q-axis current loop tuning experiment duration.

**Programmatic Use**

**Block Parameter:** DurationQaxis

**Type:** positive scalar

**Default:** 0.05

**Speed loop start time (sec) — Specify speed loop tuning experiment start time**

2 (default)

Specify the simulation time when the speed loop tuning experiment starts.

**Programmatic Use**

**Block Parameter:** StartTimeSpeed

**Type:** positive scalar

**Default:** 2

**Speed loop experiment duration (sec) — Specify speed loop tuning experiment duration**

3 (default)

Specify the speed loop tuning experiment duration.

**Programmatic Use**

**Block Parameter:** DurationSpeed

**Type:** positive scalar

**Default:** 3

**Flux loop start time (sec) — Specify flux loop tuning experiment start time**

6 (default)

Specify the simulation time when the flux tuning experiment starts.

**Programmatic Use****Block Parameter:** StartTimeFlux**Type:** positive scalar**Default:** 6**Flux loop experiment duration (sec) — Specify flux loop tuning experiment duration**  
3 (default)

Specify the flux loop tuning experiment duration.

**Programmatic Use****Block Parameter:** DurationFlux**Type:** positive scalar**Default:** 3**Loop Experiment Settings****Experiment Mode — Sinusoidal perturbation signal type**

Superposition (default) | Sinestream

Specify whether the perturbation at each frequency is applied sequentially (**Sinestream**) or simultaneously (**Superposition**).

- **Sinestream** — In this mode, the block applies perturbation at each frequency separately. For more information about sinestream signals for estimation, see “Sinestream Input Signals” (Simulink Control Design).
- **Superposition** — In this mode, the perturbation signal includes all specified frequencies at once. For frequency response estimation at a vector of frequencies  $\omega = [\omega_1, \dots, \omega_N]$  at amplitudes  $A = [A_1, \dots, A_N]$ , the perturbation signal is:

$$\Delta u = \sum_i A_i \sin(\omega_i t).$$

**Sinestream** mode can be more accurate and can also be less intrusive, because the total size of the perturbation is never bigger than the values specified by the **Sine Amplitudes** parameter. However, due to the sequential nature of the sinestream perturbation, each frequency point you add increases the recommended experiment time (see the **start/stop** input port for details). Thus, the estimation experiment is typically much faster in **Superposition** mode with satisfactory results.

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.

**Programmatic Use****Block Parameter:** ExperimentMode**Type:** character vector**Values:** 'Superposition' | 'Sinestream'**Default:** 'Superposition'**D-axis Current Loop****Plant Type — Stability of direct-axis current plant**

Stable (default) | Integrating

Specify whether the plant associated with the d-axis current control loop is stable or integrating. If the plant has one or more integrators, select **Integrating**.

**Programmatic Use****Block Parameter:** PlantTypeDaxis**Type:** character vector**Values:** 'Stable' | 'Integrating'**Default:** 'Stable'**Plant Sign — Sign of direct-axis current plant**

Positive (default) | Negative

Specify whether the plant associated with the d-axis current control loop is positive or negative. If a positive change in the plant input at the nominal operating point results in a positive change in the plant output, specify **Positive**. Otherwise, specify **Negative**. For stable plants, the sign of the plant is the sign of the plant DC gain.

**Programmatic Use****Block Parameter:** PlantSignDaxis**Type:** character vector**Values:** 'Positive' | 'Negative'**Default:** 'Positive'**Sine Amplitudes — Amplitude of sinusoidal perturbations in direct-axis current loop**

1 (default) | scalar | vector of length 5

During the experiment, the block injects a sinusoidal signal into the plant associated with the loop at the frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning. Use **Sine Amplitudes** to specify the amplitude of each of these injected signals. Specify a:

- Scalar value to inject the same amplitude at each frequency
- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$

In a typical plant with typical target bandwidth, the magnitudes of the plant responses at the experiment frequencies do not vary widely. In such cases, you can use a scalar value to apply the same magnitude perturbation at all frequencies. However, if you know that the response decays sharply over the frequency range, consider decreasing the amplitude of the lower frequency inputs and increasing the amplitude of the higher frequency inputs. It is numerically better for the estimation experiment when all the plant responses have comparable magnitudes.

The perturbation amplitudes must be:

- Large enough that the perturbation overcomes any deadband in the plant actuator and generates a response above the noise level
- Small enough to keep the plant running within the approximately linear region near the nominal operating point, and to avoid saturating the plant input or output

When **Experiment mode** is **Superposition**, the sinusoidal signals are superimposed. Thus, the perturbation can be at least as large as the sum of all amplitudes. Make sure that the largest possible perturbation is within the range of your plant actuator. Saturating the actuator can introduce errors into the estimated frequency response.

To provide the sine amplitudes by using an input port, on the Block tab, select **Use external source for sine amplitudes**.

**Tunable:** Yes

**Programmatic Use****Block Parameter:** AmpSineDaxis**Type:** scalar, vector of length 5**Default:** 1**Q-axis Current Loop****Plant Type — Stability of quadrature-axis current plant**

Stable (default) | Integrating

Specify whether the plant associated with the q-axis current control loop is stable or integrating. If the plant has one or more integrators, select `Integrating`.

**Programmatic Use****Block Parameter:** PlantTypeQaxis**Type:** character vector**Values:** 'Stable' | 'Integrating'**Default:** 'Stable'**Plant Sign — Sign of quadrature-axis current plant**

Positive (default) | Negative

Specify whether the plant associated with the q-axis current control loop is positive or negative. If a positive change in the plant input at the nominal operating point results in a positive change in the plant output, specify `Positive`. Otherwise, specify `Negative`. For stable plants, the sign of the plant is the sign of the plant DC gain.

**Programmatic Use****Block Parameter:** PlantSignQaxis**Type:** character vector**Values:** 'Positive' | 'Negative'**Default:** 'Positive'**Sine Amplitudes — Amplitude of sinusoidal perturbations in quadrature-axis current loop**

1 (default) | scalar | vector of length 5

During the experiment, the block injects a sinusoidal signal into the plant associated with the loop at the frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning. Use **Sine Amplitudes** to specify the amplitude of each of these injected signals. Specify a:

- Scalar value to inject the same amplitude at each frequency
- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$

In a typical plant with typical target bandwidth, the magnitudes of the plant responses at the experiment frequencies do not vary widely. In such cases, you can use a scalar value to apply the same magnitude perturbation at all frequencies. However, if you know that the response decays sharply over the frequency range, consider decreasing the amplitude of the lower frequency inputs and increasing the amplitude of the higher frequency inputs. It is numerically better for the estimation experiment when all the plant responses have comparable magnitudes.

The perturbation amplitudes must be:

- Large enough that the perturbation overcomes any deadband in the plant actuator and generates a response above the noise level

- Small enough to keep the plant running within the approximately linear region near the nominal operating point, and to avoid saturating the plant input or output

When **Experiment mode** is **Superposition**, the sinusoidal signals are superimposed. Thus, the perturbation can be at least as large as the sum of all amplitudes. Make sure that the largest possible perturbation is within the range of your plant actuator. Saturating the actuator can introduce errors into the estimated frequency response.

To provide the sine amplitudes by using an input port, on the Block tab, select **Use external source for sine amplitudes**.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** AmpSineQaxis

**Type:** scalar, vector of length 5

**Default:** 1

**Speed Loop**

**Plant Type — Stability of speed loop plant**

Stable (default) | Integrating

Specify whether the plant associated with the speed control loop is stable or integrating. If the plant has one or more integrators, select Integrating.

**Programmatic Use**

**Block Parameter:** PlantTypeSpeed

**Type:** character vector

**Values:** 'Stable' | 'Integrating'

**Default:** 'Stable'

**Plant Sign — Sign of speed loop plant**

Positive (default) | Negative

Specify whether the plant associated with the speed control loop is positive or negative. If a positive change in the plant input at the nominal operating point results in a positive change in the plant output, specify Positive. Otherwise, specify negative. For stable plants, the sign of the plant is the sign of the plant DC gain.

**Programmatic Use**

**Block Parameter:** PlantSignSpeed

**Type:** character vector

**Values:** 'Positive' | 'Negative'

**Default:** 'Positive'

**Sine Amplitudes — Amplitude of sinusoidal perturbations in speed loop**

1 (default) | scalar | vector of length 5

During the experiment, the block injects a sinusoidal signal into the plant associated with the loop at the frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning. Use **Sine Amplitudes** to specify the amplitude of each of these injected signals. Specify a:

- Scalar value to inject the same amplitude at each frequency
- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$

In a typical plant with typical target bandwidth, the magnitudes of the plant responses at the experiment frequencies do not vary widely. In such cases, you can use a scalar value to apply the same magnitude perturbation at all frequencies. However, if you know that the response decays sharply over the frequency range, consider decreasing the amplitude of the lower frequency inputs and increasing the amplitude of the higher frequency inputs. It is numerically better for the estimation experiment when all the plant responses have comparable magnitudes.

The perturbation amplitudes must be:

- Large enough that the perturbation overcomes any deadband in the plant actuator and generates a response above the noise level
- Small enough to keep the plant running within the approximately linear region near the nominal operating point, and to avoid saturating the plant input or output

When **Experiment mode** is **Superposition**, the sinusoidal signals are superimposed. Thus, the perturbation can be at least as large as the sum of all amplitudes. Make sure that the largest possible perturbation is within the range of your plant actuator. Saturating the actuator can introduce errors into the estimated frequency response.

To provide the sine amplitudes by using an input port, on the Block tab, select **Use external source for sine amplitudes**.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** AmpSineSpeed

**Type:** scalar, vector of length 5

**Default:** 1

**Flux Loop**

**Plant Type — Stability of flux loop plant**

Stable (default) | Integrating

Specify whether the plant associated with the flux control loop is stable or integrating. If the plant has one or more integrators, select **Integrating**.

**Programmatic Use**

**Block Parameter:** PlantTypeFlux

**Type:** character vector

**Values:** 'Stable' | 'Integrating'

**Default:** 'Stable'

**Plant Sign — Sign of flux loop plant**

Positive (default) | Negative

Specify whether the plant associated with the flux control loop is positive or negative. If a positive change in the plant input at the nominal operating point results in a positive change in the plant output, specify **Positive**. Otherwise, specify **Negative**. For stable plants, the sign of the plant is the sign of the plant DC gain.

**Programmatic Use**

**Block Parameter:** PlantSignFlux

**Type:** character vector

**Values:** 'Positive' | 'Negative'



**Default:** 'Positive'

### Sine Amplitudes — Amplitude of sinusoidal perturbations in flux loop

1 (default) | scalar | vector of length 5

During the experiment, the block injects a sinusoidal signal into the plant associated with the loop at the frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning. Use **Sine Amplitudes** to specify the amplitude of each of these injected signals. Specify a:

- Scalar value to inject the same amplitude at each frequency
- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$

In a typical plant with typical target bandwidth, the magnitudes of the plant responses at the experiment frequencies do not vary widely. In such cases, you can use a scalar value to apply the same magnitude perturbation at all frequencies. However, if you know that the response decays sharply over the frequency range, consider decreasing the amplitude of the lower frequency inputs and increasing the amplitude of the higher frequency inputs. It is numerically better for the estimation experiment when all the plant responses have comparable magnitudes.

The perturbation amplitudes must be:

- Large enough that the perturbation overcomes any deadband in the plant actuator and generates a response above the noise level
- Small enough to keep the plant running within the approximately linear region near the nominal operating point, and to avoid saturating the plant input or output

When **Experiment mode** is **Superposition**, the sinusoidal signals are superimposed. Thus, the perturbation can be at least as large as the sum of all amplitudes. Make sure that the largest possible perturbation is within the range of your plant actuator. Saturating the actuator can introduce errors into the estimated frequency response.

To provide the sine amplitudes by using an input port, on the Block tab, select **Use external source for sine amplitudes**.

**Tunable:** Yes

#### Programmatic Use

**Block Parameter:** AmpSineFlux

**Type:** scalar, vector of length 5

**Default:** 1

#### Current Loops (D-axis + Q-axis)

### Plant Type — Stability of current loop plants

Stable (default) | Integrating

Specify whether the plants associated with the current control loops are stable or integrating. If the plant has one or more integrators, select Integrating.

#### Programmatic Use

**Block Parameter:** PlantTypeAllInner

**Type:** character vector

**Values:** 'Stable' | 'Integrating'

**Default:** 'Stable'

**Plant Sign — Sign of current loop plants**

Positive (default) | Negative

Specify whether the plants associated with the current control loops are positive or negative. If a positive change in the plant input at the nominal operating point results in a positive change in the plant output, specify **Positive**. Otherwise, specify **Negative**. For stable plants, the sign of the plant is the sign of the plant DC gain.

**Programmatic Use****Block Parameter:** PlantSignAllInner**Type:** character vector**Values:** 'Positive' | 'Negative'**Default:** 'Positive'**Sine Amplitudes — Amplitude of sinusoidal perturbations in current loops**

1 (default) | scalar | vector of length 5

During the experiment, the block injects a sinusoidal signal into the plant associated with the loop at the frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning. Use **Sine Amplitudes** to specify the amplitude of each of these injected signals. Specify a:

- Scalar value to inject the same amplitude at each frequency
- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$

In a typical plant with typical target bandwidth, the magnitudes of the plant responses at the experiment frequencies do not vary widely. In such cases, you can use a scalar value to apply the same magnitude perturbation at all frequencies. However, if you know that the response decays sharply over the frequency range, consider decreasing the amplitude of the lower frequency inputs and increasing the amplitude of the higher frequency inputs. It is numerically better for the estimation experiment when all the plant responses have comparable magnitudes.

The perturbation amplitudes must be:

- Large enough that the perturbation overcomes any deadband in the plant actuator and generates a response above the noise level
- Small enough to keep the plant running within the approximately linear region near the nominal operating point, and to avoid saturating the plant input or output

When **Experiment mode** is **Superposition**, the sinusoidal signals are superimposed. Thus, the perturbation can be at least as large as the sum of all amplitudes. Make sure that the largest possible perturbation is within the range of your plant actuator. Saturating the actuator can introduce errors into the estimated frequency response.

To provide the sine amplitudes by using an input port, on the Block tab, select **Use external source for sine amplitudes**.

**Tunable:** Yes**Programmatic Use****Block Parameter:** AmpSineAllInner**Type:** scalar, vector of length 5**Default:** 1

**Outer Loops (Speed + Flux)****Plant Type — Stability of outer loop plants**

Stable (default) | Integrating

Specify whether the plants associated with the outer control loops are stable or integrating. If the plant has one or more integrators, select Integrating.

**Programmatic Use****Block Parameter:** PlantTypeAllOuter**Type:** character vector**Values:** 'Stable' | 'Integrating'**Default:** 'Stable'**Plant Sign — Sign of outer loop plants**

Positive (default) | Negative

Specify whether the plants associated with the outer control loops are positive or negative. If a positive change in the plant input at the nominal operating point results in a positive change in the plant output, specify Positive. Otherwise, specify negative. For stable plants, the sign of the plant is the sign of the plant DC gain.

**Programmatic Use****Block Parameter:** PlantSignAllOuter**Type:** character vector**Values:** 'Positive' | 'Negative'**Default:** 'Positive'**Sine Amplitudes — Amplitude of sinusoidal perturbations in outer loops**

1 (default) | scalar | vector of length 5

During the experiment, the block injects a sinusoidal signal into the plant associated with the loop at the frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ , where  $\omega_c$  is the target bandwidth for tuning. Use **Sine Amplitudes** to specify the amplitude of each of these injected signals. Specify a:

- Scalar value to inject the same amplitude at each frequency
- Vector of length 5 to specify a different amplitude at each of  $[1/10, 1/3, 1, 3, 10]\omega_c$

In a typical plant with typical target bandwidth, the magnitudes of the plant responses at the experiment frequencies do not vary widely. In such cases, you can use a scalar value to apply the same magnitude perturbation at all frequencies. However, if you know that the response decays sharply over the frequency range, consider decreasing the amplitude of the lower frequency inputs and increasing the amplitude of the higher frequency inputs. It is numerically better for the estimation experiment when all the plant responses have comparable magnitudes.

The perturbation amplitudes must be:

- Large enough that the perturbation overcomes any deadband in the plant actuator and generates a response above the noise level
- Small enough to keep the plant running within the approximately linear region near the nominal operating point, and to avoid saturating the plant input or output

When **Experiment mode** is **Superposition**, the sinusoidal signals are superimposed. Thus, the perturbation can be at least as large as the sum of all amplitudes. Make sure that the largest possible

perturbation is within the range of your plant actuator. Saturating the actuator can introduce errors into the estimated frequency response.

To provide the sine amplitudes by using an input port, on the Block tab, select **Use external source for sine amplitudes**.

**Tunable:** Yes

**Programmatic Use**

**Block Parameter:** AmpSineAllOuter

**Type:** scalar, vector of length 5

**Default:** 1

**Block Tab**

**Use external source for bandwidths — Supply external signal for target bandwidths**

off (default) | on

Select this parameter to enable the bandwidth input port of the block. You can specify the target bandwidth for all the loops the block tunes at this port. When this parameter is disabled, specify the target bandwidths at the block parameters. For more details, see the bandwidth port description.

**Programmatic Use**

**Block Parameter:** UseExternalWc

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Use external source for target phase margins — Supply external signal for target phase margin**

off (default) | on

Select this parameter to enable the target PM input port of the block. You can specify the target phase margin for all the loops the block tunes at this port. When this parameter is disabled, specify the target phase margins at the block parameters. For more details, see the target PM port description.

**Programmatic Use**

**Block Parameter:** UseExternalPM

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

**Use external source for start/stops of experiment — Supply external signal for start and stop of tuning experiment**

off (default) | on

Select this parameter to enable the start/stop and ActiveLoop input ports of the block. You can specify the start and stop of the experiment and which loop the block tunes at these ports. When this parameter is disabled, specify the start time and duration of the tuning experiment at the block parameters. For more details, see the start/stop and ActiveLoop port descriptions.

**Programmatic Use**

**Block Parameter:** UseExternalSourceStartStop

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### Use external source for sine amplitudes — Supply external signal for sinusoidal perturbation amplitude

off (default) | on

Select this parameter to enable the `sine Amp` input port of the block. You can specify sinusoidal perturbation amplitude for all the loops the block tunes at this port. When this parameter is disabled, supply the sine amplitudes at block parameters. For more details, see the `sine Amp` port description.

#### Programmatic Use

**Block Parameter:** UseExternalAmpSine

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### Data Type — Floating point precision

double (default) | single

Specify the floating-point precision based on the simulation environment or hardware requirements.

#### Programmatic Use

**Block Parameter:** BlockDataType

**Type:** character vector

**Values:** 'double' | 'single'

**Default:** 'double'

### Estimated phase margin achieved by tuned controllers — Phase margin achieved by most recently tuned loop

off (default) | on

Select this parameter to enable the `estimated PM` output port of the block. The block returns the phase margin achieved by the tuned controller of the most recently tuned loop. When this parameter is disabled, you can see the tuning results by using the **Export to MATLAB** parameter. For more details, see the `estimated PM` port description.

#### Programmatic Use

**Block Parameter:** UseExternalAchievedPM

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### Plant frequency responses near bandwidth — Estimated frequency response for most recently tuned loop

off (default) | on

Select this parameter to enable the `frd` output port of the block. The block returns the phase margin achieved by the tuned controller of the most recently tuned loop. When this parameter is disabled, you can see the tuning results by using the **Export to MATLAB** parameter. For more details, see the `frd` port description.

#### Programmatic Use

**Block Parameter:** UseExternalFRD

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### **Plant nominal input and output — Plant input and output at nominal operating point**

off (default) | on

Select this parameter to enable the nominal output port of the block. The block returns the plant input and output at the nominal operating point of the most recently tuned loop. When this parameter is disabled, you can see the tuning results by using the **Export to MATLAB** parameter. For more details, see the port description.

#### **Programmatic Use**

**Block Parameter:** UseExternalU0

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### **Start/stop of autotuning process — Signal indicating start and end of experiment for each tuned loop**

off (default) | on

Select this parameter to enable loop start/stops output port of the block. The block returns a signal indicating the times at which the autotuning experiment started and ended for each loop tuned by the block. When this parameter is disabled, you can see the tuning results by using the **Export to MATLAB** parameter. For more details, see the loop start/stops port description.

#### **Programmatic Use**

**Block Parameter:** UseExternalActiveLoop

**Type:** character vector

**Values:** 'off' | 'on'

**Default:** 'off'

### **Export to MATLAB — Send experiment and tuning results to MATLAB workspace**

button

When you click this button, the block creates a structure in the MATLAB workspace containing the experiment and tuning results. This structure, `FOCTuningResult`, contains the tuning results for each loop the block tunes.

- `Daxis` — D-axis current loop tuning results
- `Qaxis` — Q-axis current loop tuning results
- `Speed` — Speed loop tuning results
- `Flux` — Flux loop tuning results

For each loop tuned by the block, the result contains the following fields:

- `P, I, D, N` — Tuned PID gains. The structure contains whichever of these fields are necessary for the controller type you are tuning. For instance, if you are tuning a PI controller, the structure contains `P` and `I`, but not `D` and `N`.
- `TargetBandwidth` — The value you specified in the `Target bandwidth (rad/sec)` parameter of the block.
- `TargetPhaseMargin` — The value you specified in the `Target phase margin (degrees)` parameter of the block.

- `EstimatedPhaseMargin` — Estimated phase margin achieved by the tuned system.
- `Controller` — The tuned PID controller, returned as a `pid` (for parallel form) or `pidstd` (for ideal form) model object.
- `Plant` — The estimated plant, returned as an `frd` model object. This `frd` contains the response data obtained at the experiment frequencies  $[1/10, 1/3, 1, 3, 10]\omega_c$ .
- `PlantNominal` — The plant input and output at the nominal operating point when the experiment begins, specified as a structure with the fields `u` (input) and `y` (output).

You can export to the MATLAB workspace while the simulation is running, including when running in external mode.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

The generated code for this block can be resource heavy. For real-time applications, deploying the code on rapid prototyping hardware, such as the Speedgoat real-time target machine, is recommended.

### PLC Code Generation

Generate Structured Text code using Simulink® PLC Coder™.

## See Also

Park Transform | Discrete PI Controller with anti-windup and reset | DQ Limiter | Speed Measurement | Inverse Park Transform

### Topics

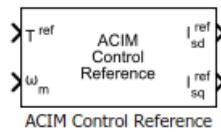
“How to Use Field Oriented Control Autotuner Block”

### Introduced in R2020a

## ACIM Control Reference

Compute reference currents for field-oriented control of induction motor

**Library:** Motor Control Blockset / Controls / Control Reference



### Description

The ACIM Control Reference block computes the  $d$ -axis and  $q$ -axis reference currents for the field-oriented control (and field-weakening) operation.

The block accepts the reference torque and feedback mechanical speed and outputs the corresponding  $d$ - and  $q$ -axes reference currents.

The block computes the reference current values by solving mathematical relationships. The calculations use the SI unit system. When working with the Per-Unit (PU) system (with the **Input units** parameter set to Per-Unit (PU)), the block converts PU input signals to SI units to perform the computations and converts them back to PU values at the output.

These equations describe how the block computes the reference  $d$ -axis and  $q$ -axis current values.

### Mathematical Model of Induction Motor

These model equations describe the dynamics of induction motor in the rotor flux reference frame:

The machine inductances are represented as,

$$L_s = L_{ls} + L_m$$

$$L_r = L_{lr} + L_m$$

$$\sigma = 1 - \left( \frac{L_m^2}{L_s \cdot L_r} \right)$$

Stator voltages are represented as,

$$v_{sd} = R_s i_{sd} + \sigma L_s \frac{di_{sd}}{dt} + \frac{L_m}{L_r} \frac{d\lambda_{rd}}{dt} - \omega_e \sigma L_s i_{sq}$$

$$v_{sq} = R_s i_{sq} + \sigma L_s \frac{di_{sq}}{dt} + \frac{L_m}{L_r} \omega_e \lambda_{rd} + \omega_e \sigma L_s i_{sd}$$

In the preceding equations, the flux linkages can be represented as,

$$\lambda_{sd} = \frac{L_m}{L_r} \lambda_{rd} + \sigma L_s i_{sd}$$

$$\lambda_{sq} = \sigma L_s i_{sq}$$



$$\tau_r \frac{d\lambda_{rd}}{dt} + \lambda_{rd} = L_m i_{sd}$$

If we keep the rotor flux as constant and the  $d$ -axis is aligned to the rotor flux reference frame, then we can imply:

$$\lambda_{rd} = L_m i_{sd}$$

$$\lambda_{rq} = 0$$

These equations describe the mechanical dynamics,

$$T_e = \frac{3}{2} p \left( \frac{L_m}{L_r} \right) \lambda_{rd} i_{sq}$$

$$T_e - T_L = J \frac{d\omega_m}{dt} + B\omega_m$$

These equations describe the slip speed,

$$\tau_r = \left( \frac{L_r}{R_r} \right)$$

$$\omega_{e\_slip} = \left( \frac{L_m \cdot i_{sq}^{ref}}{\tau_r \cdot \lambda_{rd}} \right)$$

$$\omega_e = \omega_r + \omega_{e\_slip}$$

$$\theta_e = \int \omega_e \cdot dt = \int (\omega_r + \omega_{e\_slip}) \cdot dt = \theta_r + \theta_{slip}$$

### Reference Current Computation

These equations show computation of the reference currents,

$$i_{sd\_0} = \frac{\lambda_{rd}}{L_m}$$

$$i_{sq\_req} = \frac{T^{ref}}{\frac{3}{2} p \left( \frac{L_m}{L_r} \right) \lambda_{rd}}$$

The reference currents are computed differently for operation below base speed and field weakening region,

If  $\omega_m \leq \omega_{rated}$ :

$$i_{sd\_sat} = \min(i_{sd\_0}, i_{max})$$

If  $\omega_m > \omega_{rated}$ :

$$i_{sd\_fw} = i_{sd\_0} \left( \frac{\omega_{rated}}{\omega_m} \right)$$

$$i_{sd\_sat} = \min(i_{sd\_fw}, i_{max})$$

These equations indicate the  $q$ -axis current computation,

$$i_{sq\_lim} = \sqrt{i_{max}^2 - i_{sd\_sat}^2}$$

$$i_{sq\_sat} = sat(i_{sq\_lim}, i_{sq\_req})$$

The block outputs the following values,

$$i_{sd}^{ref} = i_{sd\_sat}$$

$$i_{sq}^{ref} = i_{sq\_sat}$$

where:

- $p$  is the number of pole pairs of the motor.
- $R_s$  is the stator phase winding resistance (Ohms).
- $R_r$  is the rotor resistance referred to stator (Ohms).
- $L_{ls}$  is the stator leakage inductance (Henry).
- $L_{lr}$  is the rotor leakage inductance (Henry).
- $L_s$  is the stator inductance (Henry).
- $L_m$  is the magnetizing inductance (Henry).
- $L_r$  is the rotor inductance referred to stator (Henry).
- $\sigma$  is the total leakage factor of the induction motor.
- $\tau_r$  is the rotor time constant (sec).
- $v_{sd}$  and  $v_{sq}$  are the stator  $d$ - and  $q$ -axis voltages (Volts).
- $i_{sd}$  and  $i_{sq}$  are the stator  $d$ - and  $q$ -axis currents (Amperes).
- $i_{sd\_0}$  is the rated  $d$ -axis current of the stator also known as magnetizing current (Amperes).
- $i_{max}$  is the maximum phase current (peak) of the motor (Amperes).
- $\lambda_{sd}$  is the  $d$ -axis flux linkage of the stator (Weber).
- $\lambda_{sq}$  is the  $q$ -axis flux linkage of the stator (Weber).
- $\lambda_{rd}$  is the  $d$ -axis flux linkage of the rotor (Weber).
- $\lambda_{rq}$  is the  $q$ -axis flux linkage of the rotor (Weber).
- $\omega_{e\_slip}$  is the electrical slip speed of the rotor (Radians/ sec).
- $\omega_{slip}$  is the mechanical slip speed of the rotor (Radians/ sec).
- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $\omega_m$  is the rotor mechanical speed (Radians/ sec).
- $\omega_r$  is the rotor electrical speed (Radians/ sec).
- $\omega_{rated}$  is the rated mechanical speed of the motor (Radians/ sec).
- $T_e$  is the electromechanical torque produced by the motor (Nm).

## Ports

### Input

#### $T^{\text{ref}}$ — Reference torque value

scalar

Reference torque input value for which the block computes the reference current.

Data Types: single | double | fixed point

#### $\omega_m$ — Mechanical speed

scalar

Reference mechanical speed value for which the block computes the reference current.

Data Types: single | double | fixed point

### Output

#### $I_{sd}^{\text{ref}}$ — Reference $d$ -axis stator current

scalar

Reference  $d$ -axis stator current value.

Data Types: single | double | fixed point

#### $I_{sq}^{\text{ref}}$ — Reference $q$ -axis stator current

scalar

Reference  $q$ -axis stator current value.

Data Types: single | double | fixed point

## Parameters

#### Number of pole pairs — Number of pole pairs available in motor

2 (default) | scalar

Number of pole pairs available in the induction motor.

#### Rotor leakage inductance (H) — Leakage inductance of rotor winding

6.81e-3 (default) | scalar

Inductance due to leakage flux linked to the rotor winding (in Henry).

#### Magnetizing Inductance (H) — Magnetizing inductance of induction motor

30e-3 (default) | scalar

Inductance due to the magnetizing flux (in Henry).

#### Rated Flux (Wb) — Rated flux of motor

38.2e-3 (default) | scalar

Rated flux of the induction motor (in Weber).

**Rated Speed (rpm) — Rated speed of motor**

1150 (default) | scalar

Rated speed of the induction motor according to motor data sheet (in rpm).

**Synchronous Speed (rpm) — Synchronous speed of motor**

1500 (default) | scalar

Synchronous speed of the induction motor (in rpm).

**Max current (A) — Maximum phase current limit for motor (amperes)**

3 (default) | scalar

Maximum phase current limit for the induction motor (amperes).

**Input units — Unit of input values**

Per-Unit (PU) (default) | SI Units

Unit of the input values.

**Base Current (A) — Base current for per-unit system**

5.3611 (default) | scalar

Base current (in Amperes) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Base torque (Nm) — Base torque for per-unit system**

0.50072 (default) | scalar

Base torque (in Nm) for per-unit system. See “Per-Unit System” page for more details.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Dependencies**

To display this parameter, set **Input units** to Per-Unit (PU).

**References**

- [1] *B. Bose, Modern Power Electronics and AC Drives. Prentice Hall, 2001. ISBN-0-13-016743-6.*
- [2] *Lorenz, Robert D., Thomas Lipo, and Donald W. Novotny. "Motion control with induction motors." Proceedings of the IEEE, Vol. 82, Issue 8, August 1994, pp. 1215-1240.*
- [3] *W. Leonhard, Control of Electrical Drives, 3rd ed. Secaucus, NJ, USA:Springer-Verlag New York, Inc., 2001.*
- [4] *Briz, Fernando, Michael W. Degner, and Robert D. Lorenz. "Analysis and design of current regulators using complex vectors." IEEE Transactions on Industry Applications, Vol. 36, Issue 3, May/June 2000, pp. 817-825.*

- [5] Briz, Fernando, et al. "Current and flux regulation in field-weakening operation [of induction motors]." *IEEE Transactions on Industry Applications*, Vol. 37, Issue 1, Jan/Feb 2001, pp. 42-50.
- [6] R. M. Prasad and M. A. Mulla, "A novel position-sensorless algorithm for field oriented control of DFIG with reduced current sensors," *IEEE Trans. Sustain. Energy*, vol. 10, no. 3, pp. 1098-1108, July 2019.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

MTPA Control Reference | ACIM Slip Speed Estimator | Discrete PI Controller with anti-windup and reset | Speed Measurement

### **Topics**

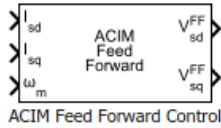
"Open-Loop and Closed-Loop Control"  
"Field-Oriented Control (FOC)"

### **Introduced in R2020b**

# ACIM Feed Forward Control

Decouple  $d$ -axis and  $q$ -axis current to eliminate disturbance

**Library:** Motor Control Blockset / Controls / Control Reference



## Description

The ACIM Feed Forward Control block decouples  $d$ -axis and  $q$ -axis current controls and generates the corresponding feed-forward voltage gains for field-oriented control of the induction motor.

The block accepts feedback values of  $d$ -axis and  $q$ -axis currents and the mechanical speed of the rotor.

## Equations

If you select Per-Unit (PU) in the **Input units** parameter, the block converts the inputs to SI units before performing any computation. After calculating the output, the block converts the values back to per-unit (PU) values.

The machine inductances and stator flux are represented as,

$$L_s = L_{ls} + L_m$$

$$L_r = L_{lr} + L_m$$

$$\sigma = 1 - \left( \frac{L_m^2}{L_s \cdot L_r} \right)$$

$$\lambda_{sd} = \frac{L_m}{L_r} \lambda_{rd} + \sigma L_s i_{sd}$$

$$\lambda_{sq} = \sigma L_s i_{sq}$$

These equations describe how the block computes the feed-forward gain.

$$V_{sd}^{FF} = \omega_e \lambda_{sd}$$

$$V_{sq}^{FF} = -\omega_e \lambda_{sq}$$

For detailed set of equations and assumptions, see “Mathematical Model of Induction Motor” on page 1-228.

where:

- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $L_{ls}$  is the stator leakage inductance (Henry).

- $L_{lr}$  is the rotor leakage inductance (Henry).
- $L_s$  is the stator inductance (Henry).
- $L_r$  is the rotor inductance (Henry).
- $L_m$  is the magnetizing inductance of the motor (Henry).
- $\sigma$  is the total leakage factor of the induction motor.
- $\lambda_{sd}$  is the  $d$ -axis flux linkage of the stator (Weber).
- $\lambda_{sq}$  is the  $q$ -axis flux linkage of the stator (Weber).
- $\lambda_{rd}$  is the  $d$ -axis flux linkage of the rotor (Weber).
- $i_{sd}$  and  $i_{sq}$  are the stator  $d$ - and  $q$ -axis currents (Amperes).

## Ports

### Input

#### $I_{sd}$ — D-axis stator current

scalar

Stator current along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

#### $I_{sq}$ — Q-axis stator current

scalar

Stator current along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

#### $\omega_m$ — Mechanical speed of rotor

scalar

Mechanical speed of the rotor.

Data Types: single | double | fixed point

### Output

#### $V_{sd}^{FF}$ — D-axis feed-forward voltage

scalar

Feed-forward voltage along the  $d$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

#### $V_{sq}^{FF}$ — Q-axis feed-forward voltage

scalar

Feed-forward voltage along the  $q$ -axis of the rotating  $dq$  reference frame.

Data Types: single | double | fixed point

## Parameters

### **Number of pole pairs — Number of pole pairs available in motor**

2 (default) | scalar

Number of pole pairs available in the induction motor.

### **Stator leakage inductance (H) — Leakage inductance of stator winding**

6.81e-3 (default) | scalar

Inductance due to leakage flux linked to the stator winding (in Henry).

### **Rotor leakage inductance (H) — Leakage inductance of rotor winding**

6.81e-3 (default) | scalar

Inductance due to leakage flux linked to the rotor winding (in Henry).

### **Magnetizing Inductance (H) — Magnetizing inductance of induction motor**

30e-3 (default) | scalar

Inductance due to the magnetizing flux (in Henry).

### **Rated Flux (Wb) — Rated flux of motor**

38.2e-3 (default) | scalar

Rated flux of the induction motor (in Weber).

### **Output Saturation (V) — Saturation limit for output values**

24/sqrt(3) (default) | scalar

Saturation limit (in Volts) for the block outputs  $V_{sd}^{FF}$  and  $V_{sq}^{FF}$ .

### **Input units — Unit of input values**

Per-Unit (PU) (default) | SI Units

Unit of the input values.

### **Base Voltage (V) — Base voltage for per-unit system**

24/sqrt(3) (default) | scalar

Base voltage (in Volts) for per-unit system.

#### **Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

### **Base Current (A) — Base current for per-unit system**

5.3611 (default) | scalar

Base current (in Amperes) for per-unit system.

#### **Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

### **Base Speed (rpm) — Base speed for per-unit system**

1500 (default) | scalar



Base speed (in rpm) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

PMSM Feed Forward Control | Park Transform | Speed Measurement | DQ Limiter | Discrete PI Controller with anti-windup and reset

**Topics**

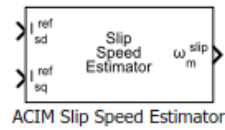
“Open-Loop and Closed-Loop Control”  
“Field-Oriented Control (FOC)”

**Introduced in R2020b**

## ACIM Slip Speed Estimator

Calculate slip speed of AC induction motor

**Library:** Motor Control Blockset / Controls / Control Reference



### Description

The ACIM Slip Speed Estimator block computes the mechanical slip speed (difference between the synchronous speed and rotor speed) of the induction motor.

The block accepts the reference values of  $d$ - and  $q$ -axis currents and outputs the computed slip speed of the induction motor.

### Equations

These equations describe the utilization of the slip speed value for field-oriented control (FOC) of the induction motor:

$$\tau_r = \left( \frac{L_r}{R_r} \right)$$

$$\omega_{e\_slip} = \left( \frac{L_m \cdot i_{sq}^{ref}}{\tau_r \cdot \lambda_{rd}} \right)$$

$$\omega_e = \omega_r + \omega_{e\_slip}$$

$$\theta_e = \int \omega_e \cdot dt = \int (\omega_r + \omega_{e\_slip}) \cdot dt = \theta_r + \theta_{slip}$$

If we keep the rotor flux as constant and the  $d$ -axis is aligned to the rotor flux reference frame, then we can imply:

$$\lambda_{rd} = L_m i_{sd}$$

This block implements the preceding calculations as:

$$\omega_{slip} = \left( \frac{1}{p} \right) \left( \frac{1}{\tau_r} \right) \left( \frac{i_{sq}^{ref}}{i_{sd}^{ref}} \right)$$

For detailed set of equations and assumptions, see “Mathematical Model of Induction Motor” on page 1-228.

where:

- $\omega_{e\_slip}$  is the electrical slip speed of the rotor (Radians/ sec).
- $\omega_{slip}$  is the mechanical slip speed of the rotor (Radians/ sec).

- $\omega_e$  is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- $\omega_m$  is the rotor mechanical speed (Radians/ sec).
- $\omega_r$  is the rotor electrical speed (Radians/ sec).
- $L_m$  is the magnetizing inductance of the motor (Henry).
- $R_r$  is the rotor resistance referred to stator (Ohms).
- $i_{sd}^{ref}$  and  $i_{sq}^{ref}$  are the reference stator  $d$ - and  $q$ -axis currents (Amperes).
- $\tau_r$  is the rotor time constant (sec).
- $\lambda_{rd}$  is the  $d$ -axis flux linkage of the rotor (Weber).

## Ports

### Input

**$I_{sd}^{ref}$  — Reference  $d$ -axis stator current**  
scalar

Reference  $d$ -axis stator current.

Data Types: single | double | fixed point

**$I_{sq}^{ref}$  — Reference  $q$ -axis stator current**  
scalar

Reference  $q$ -axis stator current.

Data Types: single | double | fixed point

### Output

**$\omega_m^{slip}$  — Slip speed of induction motor**  
scalar

Mechanical slip speed of the rotor that the block computes.

Data Types: single | double | fixed point

## Parameters

**Number of pole pairs — Number of pole pairs available in motor**  
2 (default) | scalar

Number of pole pairs available in the induction motor.

**Rotor resistance (Ohm) — Rotor resistance of motor**  
1.05 (default) | scalar

Rotor resistance of the induction motor in Ohms.

**Rotor leakage inductance (H) — Leakage inductance of rotor winding**  
6.81e-3 (default) | scalar

Inductance due to leakage flux linked to the rotor winding (in Henry).

**Magnetizing inductance (H) — Magnetizing inductance of induction motor**

30e-3 (default) | scalar

Inductance due to the magnetizing flux (in Henry).

**Output saturation (rpm) — Saturation value for the block output**

150 (default) | scalar

Saturation value for the block output (in rpm).

**Input units — Unit of input values**

Per-Unit (PU) (default) | SI Units

Unit of the input values.

**Base speed (rpm) — Base speed for per-unit system**

1500 (default) | scalar

Base speed (in rpm) for per-unit system.

**Dependencies**

To enable this parameter, set **Input units** to Per-Unit (PU).

## Extended Capabilities

**C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

Position Generator | Speed Measurement | ACIM Control Reference

**Topics**

“Open-Loop and Closed-Loop Control”

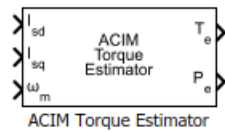
“Field-Oriented Control (FOC)”

**Introduced in R2020b**

# ACIM Torque Estimator

Estimate electromechanical torque and power

**Library:** Motor Control Blockset / Controls / Control Reference



## Description

The ACIM Torque Estimator block generates electromechanical torque and power estimates for an induction motor. The block outputs the mathematically computed electromechanical torque for constant motor parameters. To measure an accurate torque value, we recommend that you use a physical sensor.

The block accepts feedback values of  $d$ - and  $q$ -axis stator current and mechanical speed as inputs.

## Equations

If you select Per-Unit (PU) in the **Input units** parameter, the block converts the inputs to SI units before performing any computation. After calculating the output, the block converts the output back to per unit values.

These equations describe the computation of electromechanical torque and power estimates by the block.

$$T_e = \frac{3}{2}p \left( \frac{L_m}{L_r} \right) \lambda_{rd} i_{sq}$$

$$P_e = T_e \cdot \omega_m$$

For detailed set of equations and assumptions, see “Mathematical Model of Induction Motor” on page 1-228.

where:

- $p$  is the number of pole pairs available in the motor.
- $L_m$  is the magnetizing inductance of the motor (Henry).
- $L_r$  is the rotor inductance (Henry).
- $\lambda_{rd}$  is the  $d$ -axis flux linkage of the rotor (Weber).
- $i_{sq}$  is the stator  $q$ -axis current (Amperes).
- $\omega_m$  is the mechanical speed of the rotor (Radians/ sec).

## Ports

### Input

 **$I_{sd}$  — D-axis stator current**

scalar

Stator current along the  $d$ -axis of the rotating  $dq$  reference frame.Data Types: `single` | `double` | `fixed point` **$I_{sq}$  — Q-axis stator current**

scalar

Stator current along the  $q$ -axis of the rotating  $dq$  reference frame.Data Types: `single` | `double` | `fixed point` **$\omega_m$  — Mechanical speed of rotor**

scalar

Mechanical speed of the rotor.

Data Types: `single` | `double` | `fixed point`

### Output

 **$T_e$  — Electromechanical torque**

scalar

Electromechanical torque of the rotor.

Data Types: `single` | `double` | `fixed point` **$P_e$  — Electromechanical power**

scalar

Electromechanical power of the rotor.

Data Types: `single` | `double` | `fixed point`

## Parameters

**Number of pole pairs — Number of pole pairs available in motor**

2 (default) | scalar

Number of pole pairs available in the induction motor.

**Stator leakage inductance (H) — Leakage inductance of stator winding**

6.81e-3 (default) | scalar

Inductance due to leakage flux linked to the stator winding (in Henry).

**Rotor leakage inductance (H) — Leakage inductance of rotor winding**

6.81e-3 (default) | scalar

Inductance due to leakage flux linked to the rotor winding (in Henry).

**Magnetizing Inductance (H) — Magnetizing inductance of induction motor**

30e-3 (default) | scalar

Inductance due to the magnetizing flux (in Henry)..

**Rated Flux (Wb) — Rated flux of motor**

38.2e-3 (default) | scalar

Rated flux of the induction motor (in Weber).

**Input units — Unit of input values**

Per-Unit (PU) (default) | SI Units

Unit of the input values.

**Base Voltage (V) — Base voltage for per-unit system**

24/sqrt(3) (default) | scalar

Base voltage (in Volts) for per-unit system.

**Dependencies**To enable this parameter, set **Input units** to Per-Unit (PU).**Base Current (A) — Base current for per-unit system**

5.3611 (default) | scalar

Base current (in Amperes) for per-unit system.

**Dependencies**To enable this parameter, set **Input units** to Per-Unit (PU).**Base Speed (rpm) — Base speed for per-unit system**

1500 (default) | scalar

Base speed (in rpm) for per-unit system.

**Dependencies**To enable this parameter, set **Input units** to Per-Unit (PU).**Base torque (Nm) — Base torque for per-unit system**

0.50072 (default) | scalar

Base torque (in Nm) for per-unit system. See “Per-Unit System” page for more details.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Dependencies**To display this parameter, set **Input units** to Per-Unit (PU).**Base power (W) — Base power for per-unit system**

111.4284 (default) | scalar

Base power (in W) for per-unit system. See “Per-Unit System” page for more details.

This parameter is not configurable and uses a value that is internally computed using other parameters.

**Dependencies**

To display this parameter, set **Input units** to Per-Unit (PU).

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

**Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

**See Also**

Park Transform | Speed Measurement

**Topics**

“Open-Loop and Closed-Loop Control”

“Field-Oriented Control (FOC)”

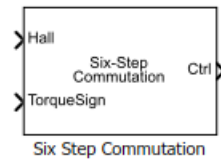
**Introduced in R2020b**



## Six Step Commutation

Generate switching sequence for six-step commutation of brushless DC (BLDC) motor

**Library:** Motor Control Blockset / Controls / Control Reference

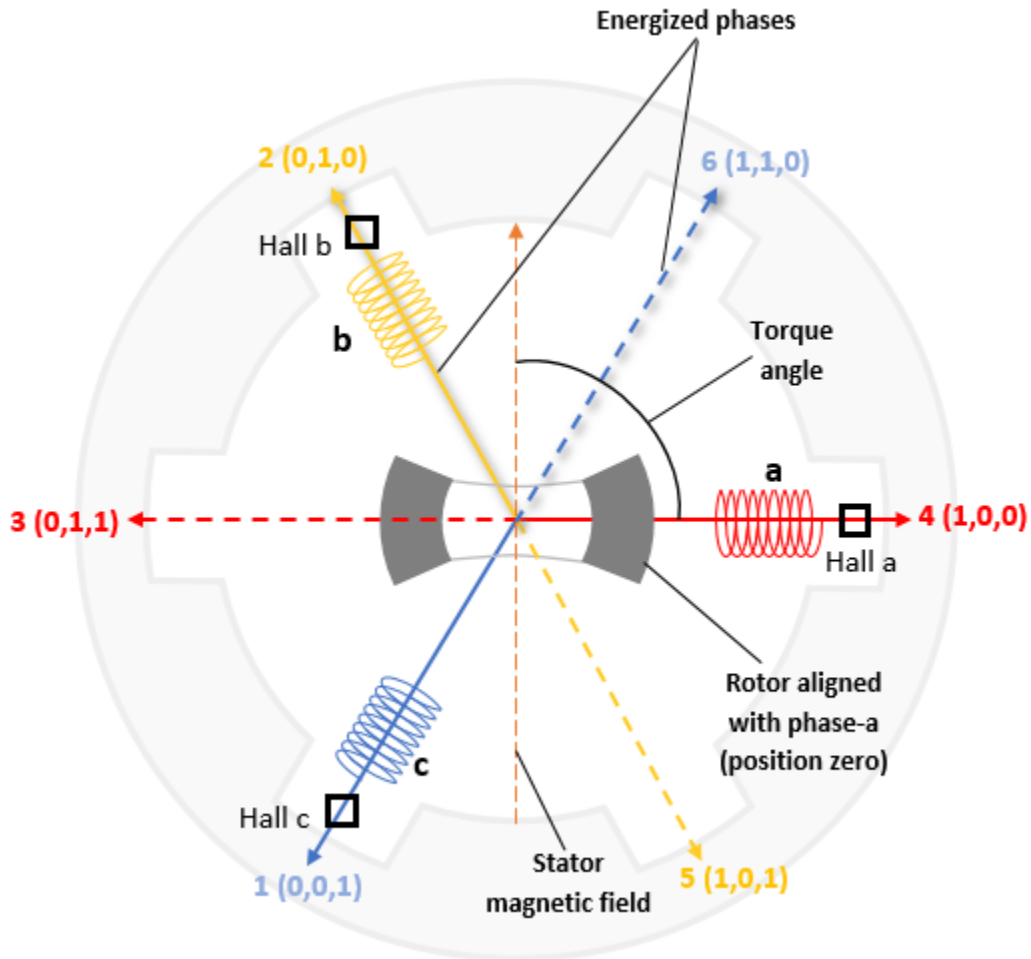


### Description

The Six Step Commutation block uses a 120-degree conduction mode to generate a switching sequence to implement six-step commutation (or trapezoidal commutation) on a three-phase BLDC motor. You can use the switching signals to operate switches and control the stator currents, and therefore, control motor speed and direction of rotation.

The block accepts the Hall sequence number or rotor position (from a position sensor such as a Hall or a quadrature encoder sensor) and the direction of torque as inputs. It uses the Hall sequence or position input to determine the sector where the rotor is present. The block computes the switching sequence such that it energizes the corresponding phases to maintain the torque angle (angle between rotor d-axis and stator magnetic field) of 90 degrees (with a deviation of 30 degrees). For example, as shown in the below figure, for hall state 5, phase B and phase C are triggered to spin the motor.

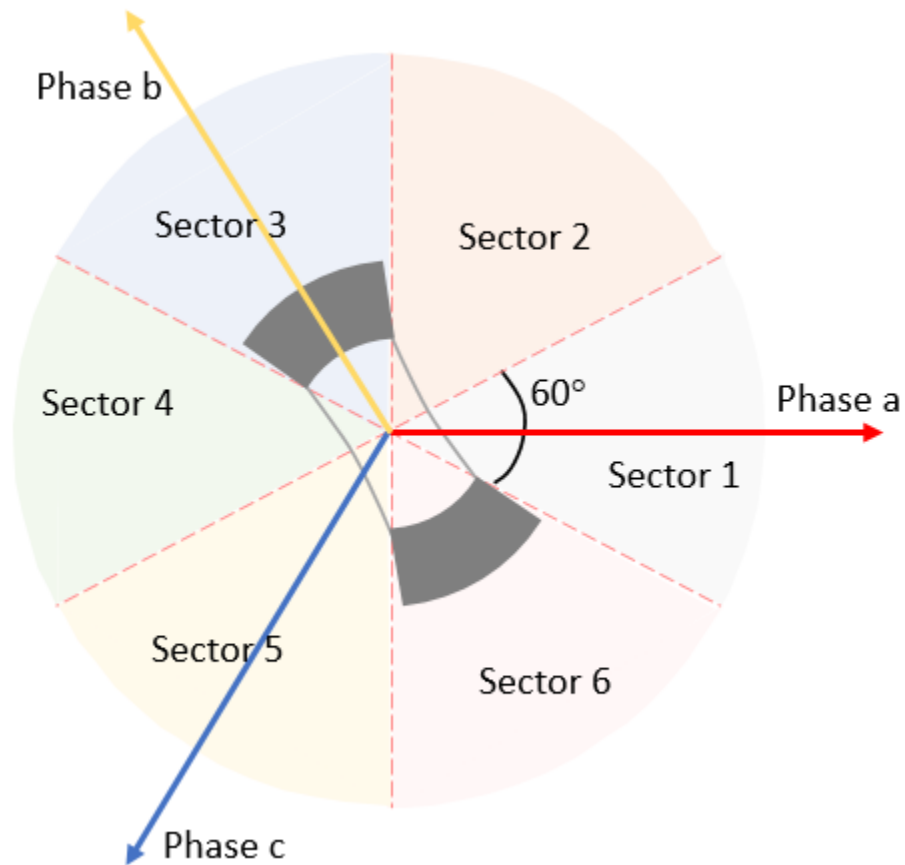
This figure is an example that shows the stator magnetic field phasors along with their default Hall sequence. It is recommended that you use "Hall Sensor Sequence Calibration of BLDC Motor" to obtain hall sequence and use this hall sequence with the block to achieve the six step commutation.



The block uses a commutation logic based on the Hall sequence to generate switching sequences.

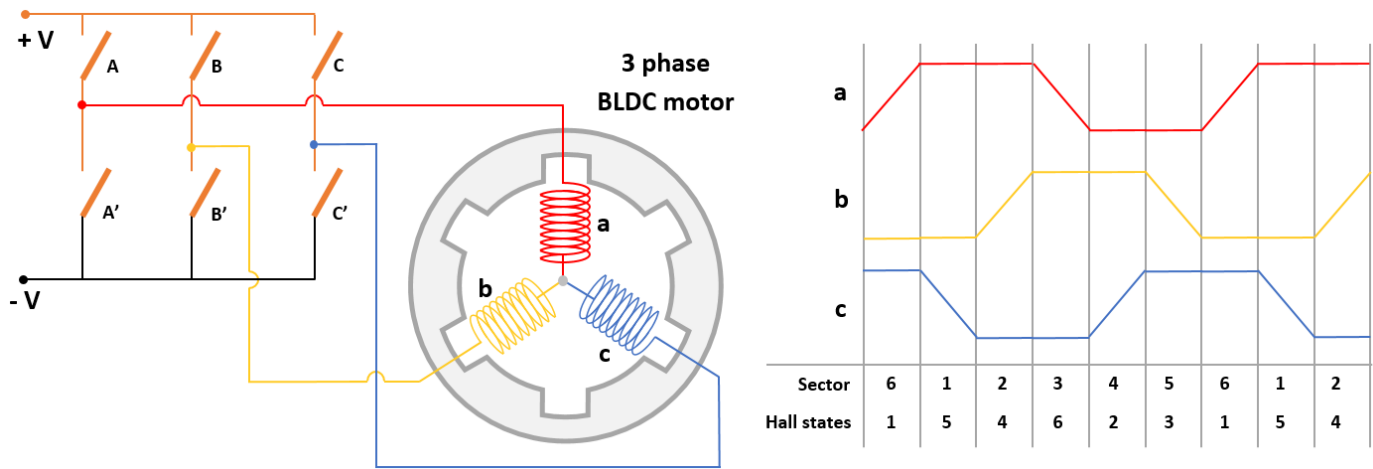
Hall State (Hall a, Hall b, Hall c)	Switching Sequence (AA' BB' CC')		
	AA'	BB'	CC'
4 (100)	00	10	01
6 (110)	01	10	00
2 (010)	01	00	10
3 (011)	00	01	10
1 (001)	10	01	00
5 (101)	10	00	01

This figure shows the stator magnetic field phasors along with the possible sectors (determined from the input rotor position).



The block uses a commutation logic based on the position sensor signals to generate switching sequences.

Position ( $\theta$ )	Sector	Switching Sequence (AA' BB' CC')		
		AA'	BB'	CC'
$(-30^\circ, 30^\circ]$	1	00	10	01
$(30^\circ, 90^\circ]$	2	01	10	00
$(90^\circ, 150^\circ]$	3	01	00	10
$(150^\circ, 210^\circ]$	4	00	01	10
$(210^\circ, 270^\circ]$	5	10	01	00
$(270^\circ, 330^\circ]$	6	10	00	01



## Ports

### Input

#### Hall — Hall sensor sequence

scalar

The Hall sensor sequence. If the Hall sensors are placed 120 degrees apart, the sequence number is between 1 to 6. For a custom Hall sensor sequence (when the Hall sensors are placed 60 degrees apart), the sequence number is between 0 to 7.

**Note** If you provide an invalid Hall sequence to this port, the block sets the output port **Ctrl** to zero.

#### Dependencies

To enable this port, set **Input type** to Hall.

Data Types: single | double | fixed point

#### Position — Rotor position

scalar

Position detected by either the Hall or quadrature encoder sensor in radians ( $0$  to  $2\pi$ ), degrees ( $0$  to  $360$ ), or per unit ( $0$  to  $1$ ).

#### Dependencies

To enable this port, set **Input type** to Position.

Data Types: single | double | fixed point

#### TorqueSign — Direction of rotation

scalar

Torque sign (+1 or -1) indicating the direction of rotation of the BLDC motor.

Data Types: single | double | fixed point

**Output****Ctrl — Motor control switching sequence**

scalar

Switching sequence signals to implement six-step commutation (or trapezoidal commutation) on the BLDC motor.

Data Types: single | double | fixed point

**Parameters****Input type — Block input type**

Hall (default) | Position

Type of position sensor feedback connected to the block input.

**Position Unit — Unit of position input**

Per-unit (default) | Degrees | Radians

Unit of position feedback input.

**Dependencies**

To enable this parameter, set **Input type** to Position.

**Hall Sequence number — Hall sequence**

[5,4,6,2,3,1] (default) | vector

Customized Hall sequence.

If the Hall sensors are placed 120 degrees apart, the sequence number is between 1 to 6. If the Hall sensors are placed 60 degrees apart, the sequence number is between 0 to 7.

**Dependencies**

To enable this parameter, set **Input type** to Hall.

**Enable custom commutation — Enable Commutation switching parameter**

off (default) | on

Select this parameter for the block to enable the **Commutation switching** parameter.

**Dependencies**

To enable this parameter, set **Input type** to Hall.

**Commutation switching — Commutation switching sequence**

[0 0 1 0 0 1;0 1 1 0 0 0;0 1 0 0 1 0;0 0 0 1 1 0;1 0 0 1 0 0;1 0 0 0 0 1] (default) | vector

Customized switching sequence for commutation of the BLDC motor.

**Dependencies**

To enable this parameter, set **Input type** to Hall and select **Enable custom commutation** parameter.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

Mechanical to Electrical Position | Discrete PI Controller with anti-windup and reset

### **Topics**

“Open-Loop and Closed-Loop Control”

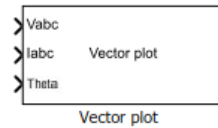
“Six-Step Commutation”

### **Introduced in R2020b**

# Vector Plot

Plot vectors in space domain

**Library:** Motor Control Blockset / Signal Management



## Description

The Vector Plot block plots and tracks the changes in vectors in the space domain. You can use the block to visualize vectors for electrical quantities (such as voltage and current) and track their changes in real time by using the trace left behind by the vector tip.

The block accepts vector magnitudes and their angles (in radians, per-unit, or degrees) as inputs and provides a pictorial representation of the vectors. The block also traces the plot history of the vector tip according to the selected number of points.

For details about how to use the Vector Plot block, see the model `mcb_pmsm_foc_qep_f28379d` in “Field-Oriented Control of PMSM Using Quadrature Encoder”.

## Ports

### Input

#### Vabc — Three-phase voltages

vector

Voltage components in the three-phase system in the *abc* reference frame. The port accepts three voltage components multiplexed by using Mux.

#### Dependencies

To enable this port, set **Select input types** to Vabc Iabc Theta.

Data Types: single | double

#### Iabc — Three-phase currents

vector

Current components in the three-phase system in the *abc* reference frame. The port accepts three current components multiplexed by using Mux.

#### Dependencies

To enable this port, set **Select input types** to Vabc Iabc Theta.

Data Types: single | double

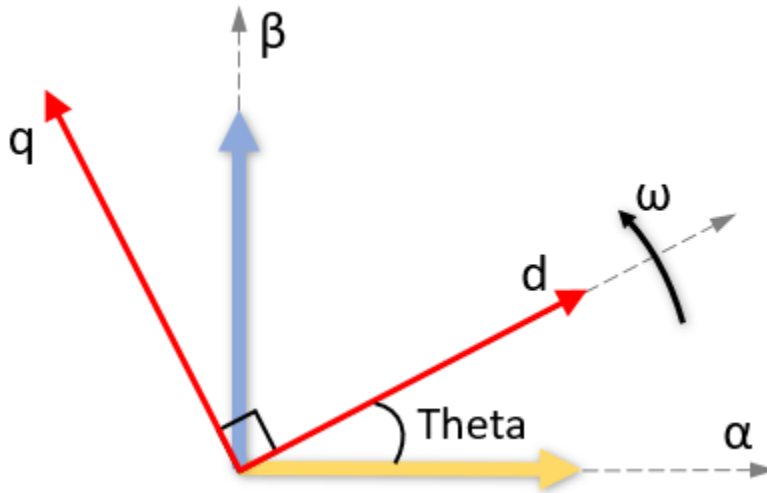
#### Theta — Angle of transformation

scalar

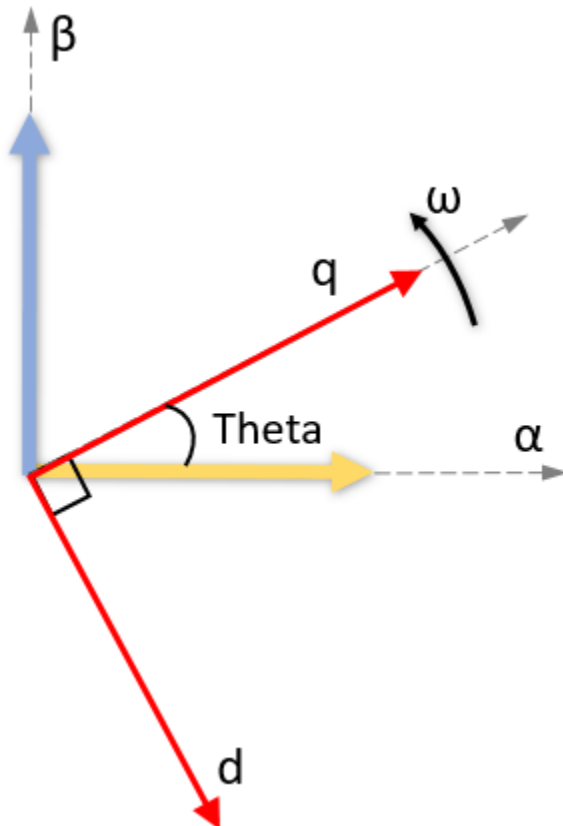
Angle value (in radians, per-unit, or degrees for  $V_{abc}$  and  $I_{abc}$ ) between the rotating reference frame and the  $\alpha$ -axis.

The figures show the angle of transformation when:

- The  $d$ -axis aligns with the  $\alpha$ -axis.



- The  $q$ -axis aligns with the  $\alpha$ -axis.





In both cases, the angle is  $\theta = \omega t$ , where:

- $\theta$  is the angle between the  $\alpha$ - and  $d$ -axes for the  $d$ -axis alignment or the angle between the  $\alpha$ - and  $q$ -axes for the  $q$ -axis alignment. It indicates the angular position of the rotating  $dq$  reference frame with respect to the  $\alpha$ -axis.
- $\omega$  is the rotational speed in the  $d$ - $q$  reference frame.
- $t$  is the time in seconds from the initial alignment.

#### Dependencies

To enable this port, set **Select input types** to `Vabc Iabc Theta` and set **Select reference frame** to `Rotating Reference Frame`.

Data Types: `single | double`

#### Vdq — Voltages in $dq$ reference frame

vector

Direct and quadrature axis voltage components in the rotating  $dq$  reference frame. The port accepts two voltage components multiplexed by using `Mux`.

#### Dependencies

To enable this port, set **Select input types** to `Vdq Idq`.

Data Types: `single | double`

#### Idq — Currents in $dq$ reference frame

vector

Direct and quadrature axis current components in the rotating  $dq$  reference frame. The port accepts two currents components multiplexed by using `Mux`.

#### Dependencies

To enable this port, set **Select input types** to `Vdq Idq`.

Data Types: `single | double`

#### Magnitude — Vector magnitudes

vector

Magnitudes of vectors that you want to plot. The port accepts up to six vector magnitudes multiplexed by using `Mux`. The vector magnitudes correspond to the angle values input to the **Angle** port.

---

**Note** The number of multiplexed vector magnitudes should be same as the number of multiplexed angles input to the **Angle** port.

---

#### Dependencies

To enable this port, set **Select input types** to `Mag Angle`.

Data Types: `single | double`

**Angle — Vector angle**

vector

Angle values (in radians, per-unit, or degrees) of vectors that you want to plot. The port accepts up to six vector angles multiplexed by using Mux. The angle values correspond to the vector magnitudes input to the **Magnitude** port.

---

**Note** The number of multiplexed vector angles should be same as the number of multiplexed magnitudes input to the **Magnitude** port.

---

**Dependencies**

To enable this port, set **Select input types** to Mag Angle.

Data Types: single | double

**Parameters****Select input types — Input port types**

Vabc Iabc Theta (default) | Vdq Idq | Mag Angle

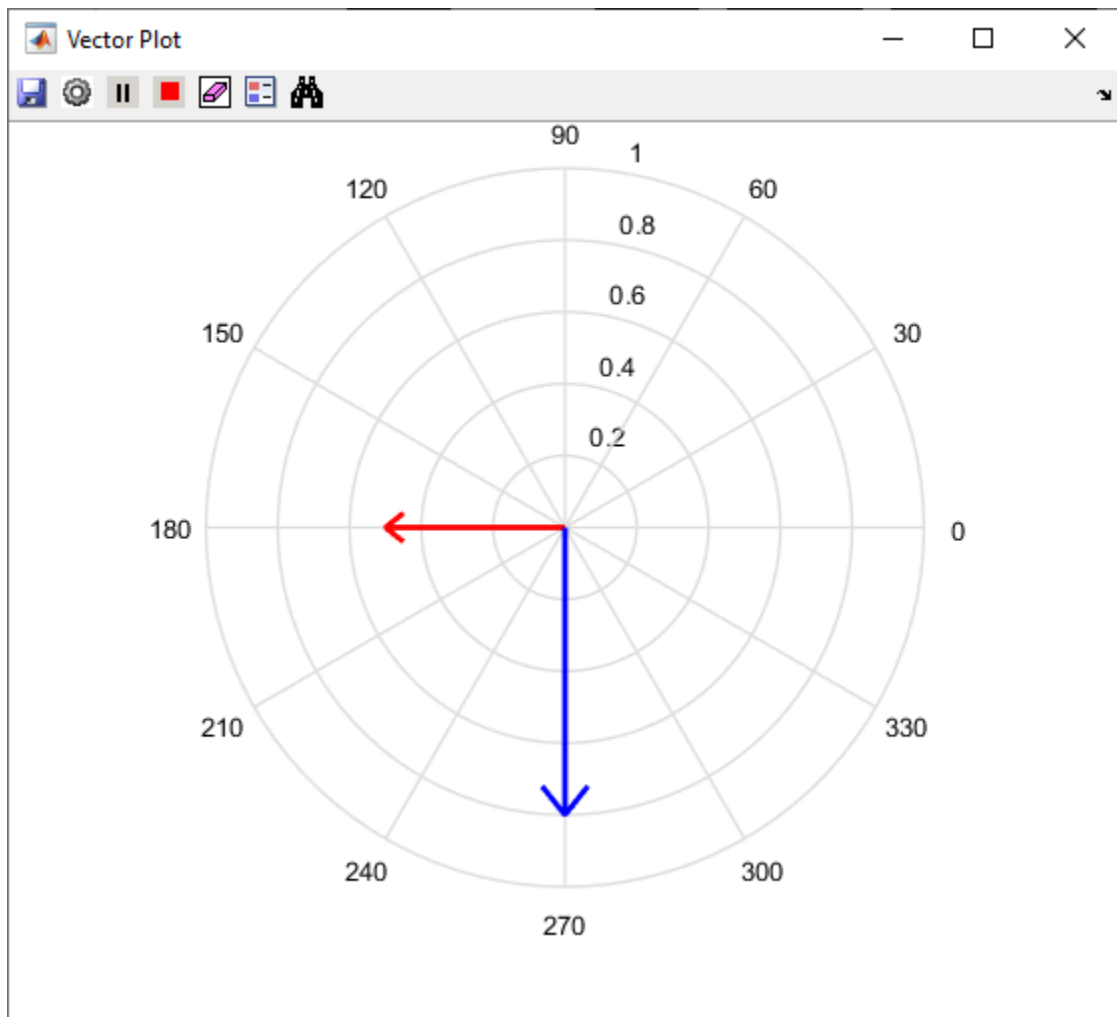
Types of input ports available for the block.

**Select reference frame — Reference frame for vectors**

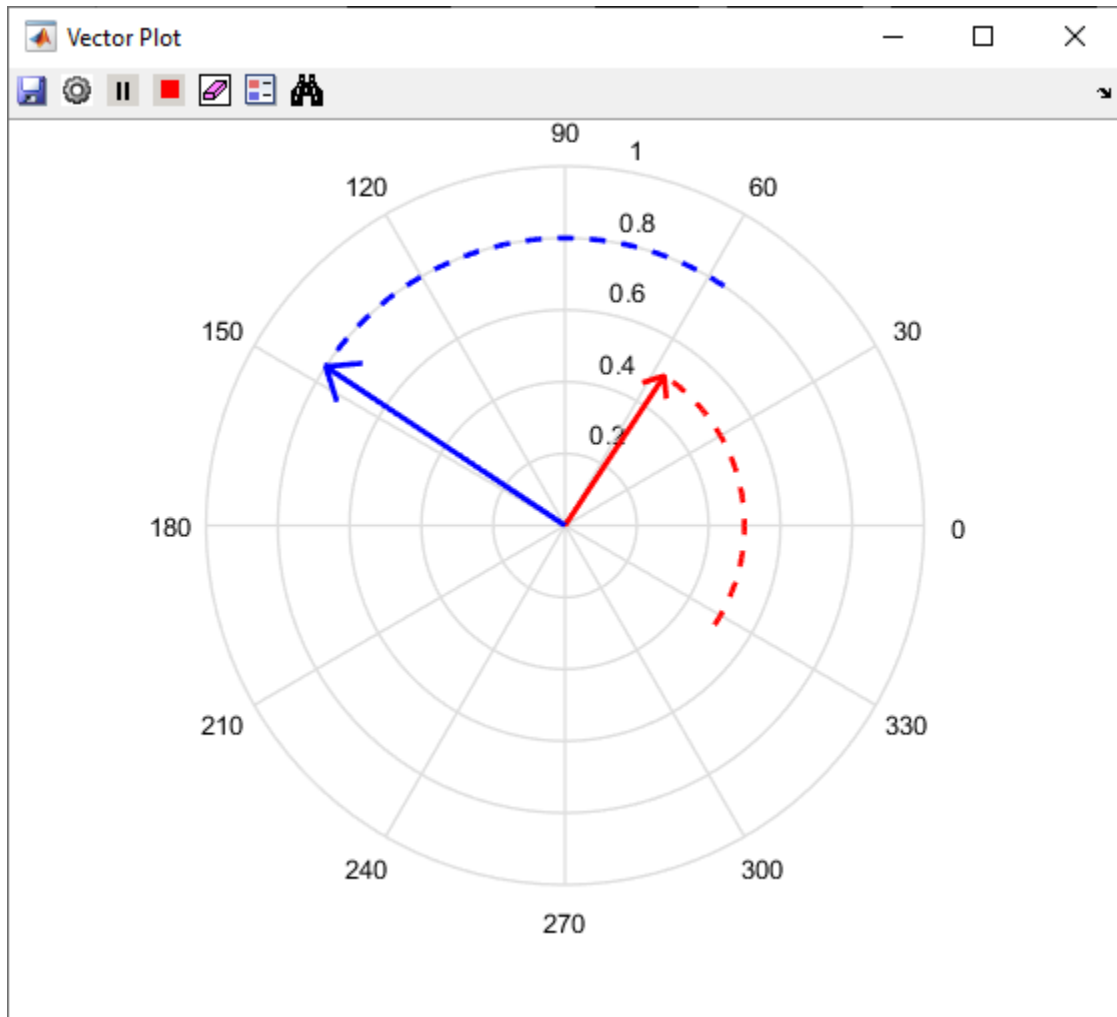
Rotating Reference Frame (default) | Stationary Reference Frame

Select type of reference frame that the block uses to plot the input vectors:

- Rotating Reference Frame — Select this option to plot the three-phase voltage and current vectors in the rotating  $dq$  reference frame.



- Stationary Reference Frame — Select this option to plot the three-phase voltage and current vectors in the stationary  $\alpha\beta$  reference frame.



### Dependencies

To enable this parameter, set **Select input types** to Vabc Iabc Theta.

### Alpha (phase-a) axis alignment — *dq* reference frame alignment

D-axis (default) | Q-axis

Align either the *d*- or *q*-axis of the rotating reference frame to the  $\alpha$ -axis of the stationary reference frame.

### Dependencies

To enable this parameter, set **Select input types** to Vabc Iabc Theta and **Select reference frame** to Rotating Reference Frame.

### Theta units — Unit of Theta input

Radians (default) | Degrees | Per-unit

Unit of **Theta** input value.

### Dependencies

To enable this parameter, set **Select input types** to Vabc Iabc Theta and **Select reference frame** to Rotating Reference Frame.

### Angle units — Unit of Angle input

Radians (default) | Degrees | Per-unit

Unit of **Angle** input value.

### Dependencies

To enable this parameter, set **Select input types** to Mag Angle.

### Open plot at simulation start — Open vector plot at simulation start

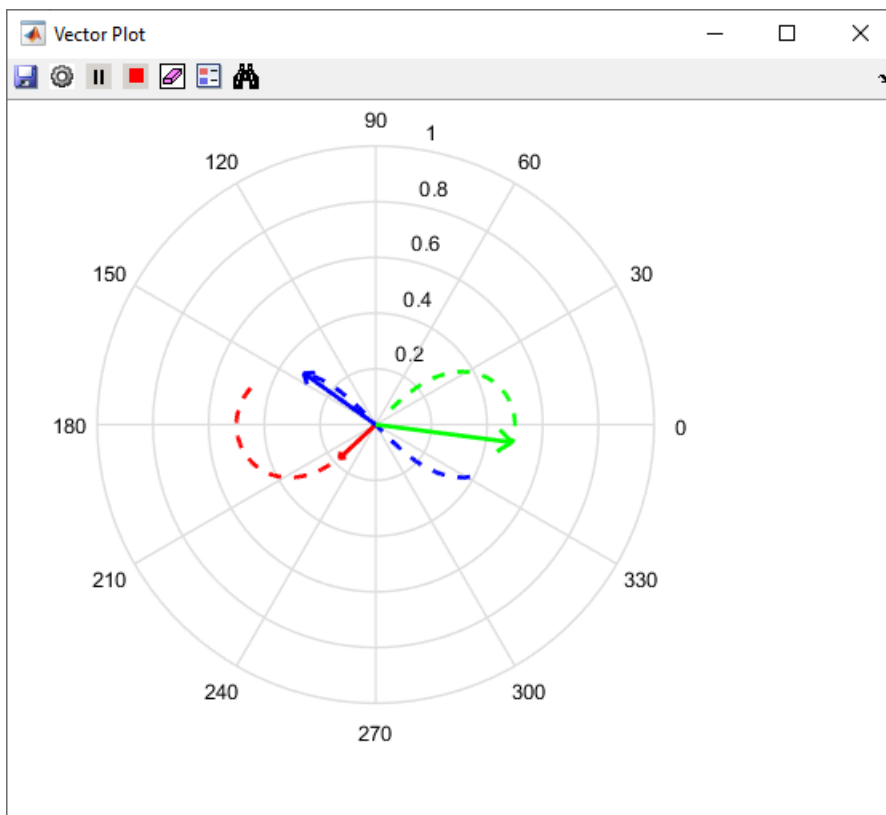
on (default) | off

Select this parameter to automatically open the vector plot window when simulation begins.








**Showplot** — Click this button to open the vector plot window.


### Vector Plot Window

This example shows the plot when you set **Select input types** to Mag Angle and provide three multiplexed magnitudes and three multiplexed angles as inputs.



You can use these buttons on the Vector Plot window:

-  (Save Figure) — Click to save the plot to an image.
-  (Preferences) — Click to open the Preferences dialog box.
  - **Display traces (samples)** — Enter the number of samples that you want to trace for the vector tip. By default, the field uses the value **100**.
  - **Auto-Scale** — Select this field to automatically scale the axes limit. The block performs auto-scaling every at every **1000** points of simulation.
  - **Axes limit** — Enter the maximum value of the  $x$  and  $y$  axis that the plot should use. By default, the field uses the value **1**. If you select **Auto-Scale** and the vector magnitude increases beyond the selected axes limit, the limit on the Vector Plot window extends automatically to accommodate the vector magnitude.
-  (Run Simulation) — Click to simulate the model that contains the Vector Plot block.
-  (Pause Simulation) — Click to pause simulation.
-  (Stop Simulation) — Click to stop simulation.
-  (Clear Data History) — Click to clear the vector tracing history.
-  (Insert Legend) — Click to insert or remove the legend describing the vectors. You can manually change the default legend description.
 

	data1
	data2
	data3
-  (Auto Scale) — Click to turn on or turn off the auto-scale function.

## Extended Capabilities

### C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

### Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

## See Also

### Topics

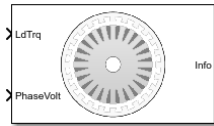
“Field-Oriented Control of PMSM Using Quadrature Encoder”

### Introduced in R2020b

# Induction Motor

Three-phase induction motor

**Library:** Powertrain Blockset / Propulsion / Electric Motors and Inverters  
Motor Control Blockset / Electrical Systems / Motors



## Description

The Induction Motor block implements a three-phase induction motor. The block uses the three-phase input voltages to regulate the individual phase currents, allowing control of the motor torque or speed.

---

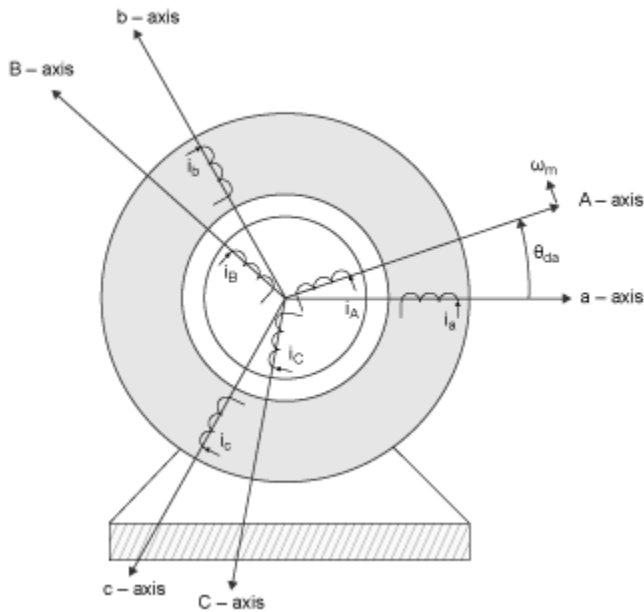
**Note** The block parameters use per-phase values of a star-equivalent induction motor.

---

By default, the block sets the **Simulation Type** parameter to **Continuous** to use a continuous sample time during simulation. If you want to generate code for fixed-step double- and single-precision targets, considering setting the parameter to **Discrete**. Then specify a **Sample Time, Ts** parameter.

### Three-Phase Sinusoidal Model Electrical System

The block implements equations that are expressed in a stationary rotor reference (qd) frame. The d-axis aligns with the a-axis. All quantities in the rotor reference frame are referred to the stator.



The block uses these equations to calculate the electrical speed ( $\omega_{em}$ ) and slip speed ( $\omega_{slip}$ ).

$$\omega_{em} = P\omega_m$$

$$\omega_{slip} = \omega_{syn} - \omega_{em}$$

To calculate the dq rotor electrical speed with respect to the rotor A-axis ( $dA$ ), the block uses the difference between the stator a-axis ( $da$ ) speed and slip speed:

$$\omega_{dA} = \omega_{da} - \omega_{em}$$

To simplify the equations for the flux, voltage, and current transformations, the block uses a stationary reference frame:

$$\omega_{da} = 0$$

$$\omega_{dA} = -\omega_{em}$$

Calculation	Equation
Flux	$\frac{d}{dt} \begin{bmatrix} \lambda_{sd} \\ \lambda_{sq} \end{bmatrix} = \begin{bmatrix} v_{sd} \\ v_{sq} \end{bmatrix} - R_s \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} - \omega_{da} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_{sd} \\ \lambda_{sq} \end{bmatrix}$ $\frac{d}{dt} \begin{bmatrix} \lambda_{rd} \\ \lambda_{rq} \end{bmatrix} = \begin{bmatrix} v_{rd} \\ v_{rq} \end{bmatrix} - R_r \begin{bmatrix} i_{rd} \\ i_{rq} \end{bmatrix} - \omega_{dA} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_{rd} \\ \lambda_{rq} \end{bmatrix}$ $\begin{bmatrix} \lambda_{sd} \\ \lambda_{sq} \\ \lambda_{rd} \\ \lambda_{rq} \end{bmatrix} = \begin{bmatrix} L_s & 0 & L_m & 0 \\ 0 & L_s & 0 & L_m \\ L_m & 0 & L_r & 0 \\ 0 & L_m & 0 & L_r \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \\ i_{rd} \\ i_{rq} \end{bmatrix}$



Calculation	Equation
Current	$\begin{bmatrix} i_{sd} \\ i_{sq} \\ i_{rd} \\ i_{rq} \end{bmatrix} = \left( \frac{1}{L_m^2 - L_r L_s} \right) \begin{bmatrix} -L_r & 0 & L_m & 0 \\ 0 & -L_r & 0 & L_m \\ L_m & 0 & -L_s & 0 \\ 0 & L_m & 0 & -L_s \end{bmatrix} \begin{bmatrix} \lambda_{sd} \\ \lambda_{sq} \\ \lambda_{rd} \\ \lambda_{rq} \end{bmatrix}$
Inductance	$L_s = L_{ls} + L_m$ $L_r = L_{lr} + L_m$
Electromagnetic torque	$T_e = PL_m(i_{sq}i_{rd} - i_{sd}i_{rq})$
Power invariant dq transformation to ensure that the dq and three phase powers are equal	$\begin{bmatrix} v_{sd} \\ v_{sq} \end{bmatrix} = \sqrt{\frac{2}{3}}$ $\begin{bmatrix} \cos(\theta_{da}) & \cos(\theta_{da} - \frac{2\pi}{3}) & \cos(\theta_{da} + \frac{2\pi}{3}) \\ -\sin(\theta_{da}) & -\sin(\theta_{da} - \frac{2\pi}{3}) & -\sin(\theta_{da} + \frac{2\pi}{3}) \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}$ $\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta_{da}) & -\sin(\theta_{da}) \\ \cos(\theta_{da} - \frac{2\pi}{3}) & -\sin(\theta_{da} - \frac{2\pi}{3}) \\ \cos(\theta_{da} + \frac{2\pi}{3}) & -\sin(\theta_{da} + \frac{2\pi}{3}) \end{bmatrix} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix}$

The equations use these variables.

$\omega_m$	Angular velocity of the rotor (rad/s)
$\omega_{em}$	Electrical rotor speed (rad/s)
$\omega_{slip}$	Electrical rotor slip speed (rad/s)
$\omega_{syn}$	Synchronous rotor speed (rad/s)
$\omega_{da}$	dq stator electrical speed with respect to the rotor a-axis (rad/s)
$\omega_{dA}$	dq stator electrical speed with respect to the rotor A-axis (rad/s)
$\theta_{da}$	dq stator electrical angle with respect to the rotor a-axis (rad)
$\theta_{dA}$	dq stator electrical angle with respect to the rotor A-axis (rad)
$L_q, L_d$	q- and d-axis inductances (H)
$L_s$	Stator inductance (H)
$L_r$	Rotor inductance (H)
$L_m$	Magnetizing inductance (H)
$L_{ls}$	Stator leakage inductance (H)
$L_{lr}$	Rotor leakage inductance (H)
$v_{sq}, v_{sd}$	Stator q- and d-axis voltages (V)
$i_{sq}, i_{sd}$	Stator q- and d-axis currents (A)
$\lambda_{sq}, \lambda_{sd}$	Stator q- and d-axis flux (Wb)
$i_{rq}, i_{rd}$	Rotor q- and d-axis currents (A)

$\lambda_{rq}, \lambda_{rd}$	Rotor q- and d-axis flux (Wb)
$v_a, v_b, v_c$	Stator voltage phases a, b, c (V)
$i_a, i_b, i_c$	Stator currents phases a, b, c (A)
$R_s$	Resistance of the stator windings (Ohm)
$R_r$	Resistance of the rotor windings (Ohm)
$P$	Number of pole pairs
$T_e$	Electromagnetic torque (Nm)

### Mechanical System

The motor angular velocity is given by:

$$\frac{d}{dt}\omega_m = \frac{1}{J}(T_e - T_f - F\omega_m - T_m)$$

$$\frac{d\theta_m}{dt} = \omega_m$$

The equations use these variables.

$J$	Combined inertia of motor and load (kgm <sup>2</sup> )
$F$	Combined viscous friction of motor and load (N·m/(rad/s))
$\theta_m$	Motor mechanical angular position (rad)
$T_m$	Motor shaft torque (Nm)
$T_e$	Electromagnetic torque (Nm)
$T_f$	Motor shaft static friction torque (Nm)
$\omega_m$	Angular mechanical velocity of the motor (rad/s)

### Power Accounting

For the power accounting, the block implements these equations.

Bus Signal		Description	Variable	Equations
PwrInfo	PwrTrnsfrd — Power transferred between blocks	PwrMtr	Mechanical power	$P_{mot} = -\omega_m T_e$
	<ul style="list-style-type: none"> <li>Positive signals indicate flow into block</li> <li>Negative signals indicate flow out of block</li> </ul>	PwrBus	Electrical power	$P_{bus} = v_a i_a + v_b i_b + v_c i_c$
	PwrNotTrnsfrd — Power crossing the block boundary, but not transferred	PwrElecLoss	Resistive power loss	$P_{elec} = -(R_s i_{sd}^2 + R_s i_{sq}^2 + R_r i_{rd}^2 + R_r i_{rq}^2)$
	<ul style="list-style-type: none"> <li>Positive signals indicate an input</li> </ul>			

Bus Signal		Description	Variable	Equations
	<ul style="list-style-type: none"> <li>Negative signals indicate a loss</li> </ul>	PwrMech Loss	Mechanical power loss	$P_{mech}$ When <b>Port Configuration</b> is set to Torque: $P_{mech} = -(\omega_m^2 F +  \omega_m  T_f)$ When <b>Port Configuration</b> is set to Speed: $P_{mech} = 0$
	PwrStored — Stored energy rate of change <ul style="list-style-type: none"> <li>Positive signals indicate an increase</li> <li>Negative signals indicate a decrease</li> </ul>	PwrMtrStored	Stored motor power	$P_{str} = P_{bus} + P_{mot} + P_{elec} + P_{mech}$

The equations use these variables.

$R_s$	Stator resistance (Ohm)
$R_r$	Motor resistance (Ohm)
$i_a, i_b, i_c$	Stator phase a, b, and c current (A)
$i_{sq}, i_{sd}$	Stator q- and d-axis currents (A)
$v_{an}, v_{bn}, v_{cn}$	Stator phase a, b, and c voltage (V)
$\omega_m$	Angular mechanical velocity of the rotor (rad/s)
$F$	Combined motor and load viscous damping (N·m/(rad/s))
$T_e$	Electromagnetic torque (Nm)
$T_f$	Combined motor and load friction torque (Nm)

## Ports

### Input

#### LdTrq — Load torque on motor

scalar

Load torque on the motor shaft,  $T_m$ , in N·m.

#### Dependencies

To create this port, select Torque for the **Port configuration** parameter.

#### Spd — Rotor shaft speed

scalar

Angular velocity of the rotor,  $\omega_m$ , in rad/s.

**Dependencies**

To create this port, select Speed for the **Port configuration** parameter.

**PhaseVolt – Stator terminal voltages**

1-by-3 array

Stator terminal voltages,  $V_a$ ,  $V_b$ , and  $V_c$ , in V.

**Output**

**Info – Bus signal**

bus

The bus signal contains these block calculations.

Signal		Description	Variable	Units	
IaStator		Stator phase current A	$i_a$	A	
IbStator		Stator phase current B	$i_b$	A	
IcStator		Stator phase current C	$i_c$	A	
IdSync		Direct axis current	$i_d$	A	
IqSync		Quadrature axis current	$i_q$	A	
VdSync		Direct axis voltage	$v_d$	V	
VqSync		Quadrature axis voltage	$v_q$	V	
MtrSpd		Angular mechanical velocity of the rotor	$\omega_m$	rad/s	
MtrMechPos		Rotor mechanical angular position	$\theta_m$	rad	
MtrPos		Rotor electrical angular position	$\theta_e$	rad	
MtrTrq		Electromagnetic torque	$T_e$	N·m	
PwrInfo	PwrTrnsfrd	PwrMtr	Mechanical power	$P_{mot}$	W
		PwrBus	Electrical power	$P_{bus}$	W
	PwrNotTrnsfrd	PwrElecLoss	Resistive power loss	$P_{elec}$	W
		PwrMechLoss	Mechanical power loss	$P_{mech}$	W
	PwrStored	PwrMtrStored	Stored motor power	$P_{str}$	W

**PhaseCurr – Phase a, b, c current**

1-by-3 array

Phase a, b, c current,  $i_a$ ,  $i_b$ , and  $i_c$ , in A.

**MtrTrq – Motor torque**

scalar

Motor torque,  $T_{mtr}$ , in N·m.

**Dependencies**

To create this port, select Speed for the **Port configuration** parameter.

**MtrSpd — Motor speed**

scalar

Angular speed of the motor,  $\omega_{mtr}$  in rad/s.

**Dependencies**

To create this port, select Torque for the **Port configuration** parameter.

**Parameters****Block Options****Simulation type — Select simulation type**

Continuous (default) | Discrete

By default, the block uses a continuous sample time during simulation. If you want to generate code for single-precision targets, considering setting the parameter to **Discrete**.

**Dependencies**

Setting **Simulation Type** to **Discrete** creates the **Sample Time, Ts** parameter.

**Sample time, Ts — Sample time for discrete integration**

0.001 (default) | scalar

Integration sample time for discrete simulation, in s.

**Dependencies**

Setting **Simulation Type** to **Discrete** creates the **Sample Time, Ts** parameter.

**Port configuration — Select port configuration**

Torque (default) | Speed

This table summarizes the port configurations.

Port Configuration	Creates Input Port	Creates Output Port
Torque	LdTrq	MtrSpd
Speed	Spd	MtrTrq

**Load Parameter Values****File — Path to motor parameter ".m" or ".mat" file**

scalar

Enter the path to the motor parameter ".m" or ".mat" file that you saved using the Motor Control Blockset parameter estimation tool. You can also click the **Browse** button to navigate and select the ".m" or ".mat" file, and update **File** parameter with the file name and path. For details related to the motor parameter estimation process, see "Estimate PMSM Parameters Using Recommended Hardware".

- **Load from file** - Click this button to read the estimated motor parameters from the ".m" or ".mat" file (indicated by the **File** parameter) and load them to the motor block.
- **Save to file** - Click this button to read the motor parameters from the motor block and save them into a ".m" or ".mat" file (with a file name and location that you specify in the **File** parameter).

---

**Note** Before you click **Save to file** button, ensure that the target file name in the **File** parameter has either ".m" or ".mat" extension. If you use any other file extension, the block displays an error message.

---

### Parameters

#### Number of pole pairs, P – Pole pairs

2 (default) | scalar

Motor pole pairs,  $P$ .

#### Stator resistance and leakage inductance, Zs – Resistance and inductance

[1.77 0.0139] (default) | vector

Stator resistance,  $R_s$ , in ohms and leakage inductance,  $L_{ls}$ , in H.

#### Rotor resistance and leakage inductance, Zr – Resistance and inductance

[1.34 0.0121] (default) | vector

Rotor resistance,  $R_r$ , in ohms and leakage inductance,  $L_{lr}$ , in H.

#### Magnetizing inductance, Lm – Inductance

0.3687 (default) | scalar

Magnetizing inductance,  $L_m$ , in H.

#### Physical inertia, viscous damping, static friction, mechanical – Inertia, damping, friction

[0.001 0 0] (default) | vector

Mechanical properties of the rotor:

- Inertia,  $J$ , in  $\text{kg}\cdot\text{m}^2$
- Viscous damping,  $F$ , in  $\text{N}\cdot\text{m}/(\text{rad}/\text{s})$
- Static friction,  $T_f$ , in  $\text{N}\cdot\text{m}$

### Dependencies

To enable this parameter, select Torque for the **Port configuration**.

### Initial Values

#### Initial mechanical position, theta\_init – Angular position

0 (default) | scalar

Initial rotor angular position,  $\theta_{m0}$ , in rad.

#### Initial mechanical speed, omega\_init – Angular speed

0 (default) | scalar

Initial angular velocity of the rotor,  $\omega_{m0}$ , in rad/s.

**Dependencies**

To enable this parameter, select Torque for the **Port configuration**.

**References**

[1] Mohan, Ned. *Advanced Electric Drives: Analysis, Control and Modeling Using Simulink*. Minneapolis, MN: MNPETE, 2001.

**Extended Capabilities****C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

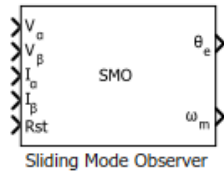
**See Also**

**Introduced in R2020b**

## Sliding Mode Observer

Compute electrical position and mechanical speed of a surface-mount PMSM

**Library:** Motor Control Blockset / Sensorless Estimators



### Description

The Sliding Mode Observer block computes the electrical position and mechanical speed of a Surface Mount PMSM by using the voltage and current values along the  $\alpha$ - and  $\beta$ -axes of the stationary  $\alpha\beta$  reference frame.

### Equations

These equations describe the discrete-time operation of a PMSM:

$$i_{\alpha\beta(k+1)} = Ai_{\alpha\beta(k)} + Bv_{\alpha\beta(k)} - Be_{\alpha\beta(k)}$$

$$e_{\alpha\beta(k+1)} = e_{\alpha\beta(k)} + T_s\omega_e(k)Je_{\alpha\beta(k)}$$

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$\Phi = \begin{bmatrix} -\frac{R}{L} & 0 \\ 0 & -\frac{R}{L} \end{bmatrix}$$

$$A = e^{\Phi T_s}$$

$$B = \int_0^{T_s} e^{\Phi\tau} d\tau = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix}$$

$$b = \frac{1 - e^{-RT_s/L}}{R}$$

These equations describe the discrete-time sliding mode observer operation of a surface mount PMSM:

$$\hat{i}_{\alpha\beta(k+1)} = A\hat{i}_{\alpha\beta(k)} + Bv_{\alpha\beta(k)} - B\hat{e}_{\alpha\beta(k)} - \eta \text{Sign}(\tilde{i}_{\alpha\beta(k)})$$

$$\hat{e}_{\alpha\beta(k+1)} = \hat{e}_{\alpha\beta(k)} + B^{-1}g(\tilde{i}_{\alpha\beta(k)} - A\tilde{i}_{\alpha\beta(k-1)} + \eta \text{Sign}(\tilde{i}_{\alpha\beta(k-1)}))$$

$$\tilde{i}_{\alpha\beta(k)} = \hat{i}_{\alpha\beta(k)} - i_{\alpha\beta(k)}$$



$$\tilde{e}_{\alpha\beta(k)} = \hat{e}_{\alpha\beta(k)} - e_{\alpha\beta(k)}$$

If the back EMF observer fulfils the conditions  $|e_{\alpha\beta(k+1)} - e_{\alpha\beta(k)}| \leq m$  and  $g \in (0, 1)$ , there exists a  $k_0$ , such that:

$$\tilde{e}_{\alpha\beta(k)} < \frac{m}{g}$$

If the sliding mode observer fulfils these conditions:

- $g \in (0, 1)$
- $|e_{\alpha\beta(k+1)} - e_{\alpha\beta(k)}| \leq m$
- $\eta > b \frac{m}{g}$

then there exists a  $k=k_0$ , such that for  $k \geq k_0$ :

$$|\tilde{i}_{\alpha\beta(k)}| \leq \eta + b \frac{m}{g}$$

where:

- $e_\alpha$  and  $i_\alpha$  are the stator back EMF and current for the  $\alpha$  axis
- $e_\beta$  and  $i_\beta$  are the stator back EMF and current for the  $\beta$  axis
- $\tilde{e}_\alpha$  and  $\tilde{i}_\alpha$  are the errors in the stator back EMF and current for the  $\alpha$  axis
- $\tilde{e}_\beta$  and  $\tilde{i}_\beta$  are the errors in the stator back EMF and current for the  $\beta$  axis
- $v_\alpha$  and  $v_\beta$  are the stator supply voltages
- $R$  is the stator resistance
- $L$  is the stator inductance
- $g$  is the back EMF observer gain
- $\eta$  is the current observer gain
- $\omega_e$  is the electrical angular velocity
- $T_s$  is the sampling period
- $k$  is the sample count

### Tuning

Use these steps to tune the block using the **Current observer gain** ( $\eta$ ) and **Back-emf observer gain** ( $g$ ) parameters.

- Select a back-emf observer gain ( $g$ ) value such that  $g \in (0, 1)$ . Bringing  $g$  close to the value 1, results in less error in the estimated back-emf. However, this makes convergence slow.
- Select a value of  $m$  based on the block sample time and maximum slope of the operating back-emf (such that  $|e_{\alpha\beta(k+1)} - e_{\alpha\beta(k)}| \leq m$ ).
- Select a current observer gain ( $\eta$ ) value based on  $b$ ,  $m$ , and  $g$  (such that  $\eta > b \frac{m}{g}$ ).

---

**Note** The block functions correctly when you tune the sliding mode observer gains.

When using open-loop control to run a motor, compute the rotor position using both sliding mode observer and an actual sensor hardware and compare the computed position values. If the difference is acceptable, the block functions correctly. Otherwise, manually tune the sliding mode observer gains to ensure that the block functions accurately.

The transition from open-loop control to closed-loop control may fail due to noise in the currents and voltages. To make a successful transition, try reducing the value of the **Filter cut-off frequency (Hz)** parameter.

---

## Ports

### Input

#### $V_\alpha$ — $\alpha$ -axis voltage

scalar

Voltage component along the  $\alpha$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### $V_\beta$ — $\beta$ -axis voltage

scalar

Voltage component along the  $\beta$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### $I_\alpha$ — $\alpha$ -axis current

scalar

Current component along the  $\alpha$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### $I_\beta$ — $\beta$ -axis current

scalar

Current component along the  $\beta$ -axis of the stationary  $\alpha\beta$  reference frame.

Data Types: `single` | `double` | `fixed point`

#### **Rst** — Reset the block

scalar

The pulse (true value) that resets and restarts the processing of the block algorithm.

Data Types: `single` | `double` | `fixed point`

### Output

#### $\theta_e$ — Electrical position of PMSM

scalar

The estimated electrical position of the rotor.

Data Types: `single` | `double` | `fixed point`

**$\omega_m$  — Mechanical speed of PMSM**

scalar

The estimated mechanical speed of the rotor.

Data Types: single | double | fixed point

**Parameters****Input units — Unit of block inputs**

SI unit (default) | Per-unit

Unit of the input voltage and current components along the  $\alpha$ -axis and  $\beta$ -axis of the stationary  $\alpha\beta$  reference frame.**Discrete step size (s) — Sample time after which block executes again**

50e-6 (default) | scalar

The fixed time interval (in seconds) between every two consecutive instances of block execution.

**Motor parameters****Stator resistance (ohm) — Resistance**

0.36 (default) | scalar

Stator phase winding resistance (in ohm).

**Stator inductance (H) — Inductance**

0.2e-3 (default) | scalar

Stator phase winding inductance (in Henry).

**Maximum application speed (RPM) — Maximum supported speed**

6000 (default) | scalar

Maximum speed (in RPM) that the block can support. For a speed beyond this value, the block generates incorrect outputs.

**Number of pole pairs — Number of pole pairs available in motor**

4 (default) | scalar

Number of pole pairs available in the motor.

**Base voltage (V) — Nominal voltage corresponding to one per unit**

13.8564 (default) | scalar

The maximum phase voltage applied to the PMSM. For details, see “Per-Unit System”.

**Base current (A) — Nominal current corresponding to one per unit**

21.4286 (default) | scalar

The maximum measurable current supplied to the PMSM. For details, see “Per-Unit System”.

---

**Note** The Sliding Mode Observer block might occasionally display the warning message 'Wrap on overflow detected.'

---

## Observer Parameters

### Back-emf observer gain — Sliding mode observer gain for back-emf

0.9 (default) | scalar

The gain that ensures the convergence of the back-emf observer.

### Current observer gain — Sliding mode observer gain for current

0.50881 (default) | scalar

The gain that ensures the convergence of the current observer.

### Filter cut-off frequency (Hz) — Cut-off frequency of internal filter

1200 (default) | scalar

The cut-off frequency of the internal low-pass IIR filter. The cut-off frequency value must be greater than or equal to the maximum electrical frequency.

Click **Compute default parameters** to calculate approximate observer gains and the filter coefficient and update these fields. For this calculation, we set  $g$  to 0.9, computed  $m$  at twice the rated speed, and set  $\eta$  to  $1.1\left(b\frac{m}{g}\right)$ .

## Datatypes

### Position unit — Unit of position output

Radians (default) | Degrees | Per unit

Unit of the position output.

### Position data type — Data type of position output

single (default) | double | fixdt(1,16) | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

Data type of the position output.

### Speed unit — Unit of speed output

RPM (default) | Degrees/sec | Radians/sec | Per unit

Unit of the speed output.

### Speed data type — Data type of speed output

single (default) | double | fixdt(1,16) | fixdt(1,16,0) | fixdt(1,16,2^0,0) | <data type expression>

Data type of the speed output.

## References

- [1] T. Bernardes, V. F. Montagner, H. A. Gründling, and H. Pinheiro, "Discrete-Time Sliding Mode Observer for Sensorless Vector Control of Permanent Magnet Synchronous Machine," in *IEEE Transactions on Industrial Electronics*, vol. 61, no. 4, pp. 1679-1691, 2014
- [2] B. Bose, *Modern Power Electronics and AC Drives*. Prentice Hall, 2001. ISBN-0-13-016743-6.

[3] J. Liu and X. Wan, "Advanced Sliding Mode Control for Mechanical Systems". Springer-Verlag Berlin Heidelberg, 2011.

## **Extended Capabilities**

### **C/C++ Code Generation**

Generate C and C++ code using Simulink® Coder™.

### **Fixed-Point Conversion**

Design and simulate fixed-point systems using Fixed-Point Designer™.

## **See Also**

Flux Observer | Clarke Transform | Inverse Park Transform | Sine-Cosine Lookup | Discrete PI Controller with anti-windup and reset

## **Topics**

“Open-Loop and Closed-Loop Control”

“Field-Oriented Control (FOC)”

## **Introduced in R2021b**

