MotorSensorless Library Documentation

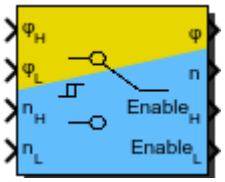
X2C v6.3.2134

February 24, 2021

Contents

1 MotorSensorless

Block: EstimatorSwitch



Inports	
phi_H	Estimated rotor angle from algorithm for high speeds
phi_L	Estimated rotor angle from algorithm for low speeds
n_H	Estimated rotor speed from algorithm for high speeds
n_L	Estimated rotor speed from algorithm for low speeds

Outports	
phi	Used estimated rotor angle
n	Used estimated rotor speed
Enable_H	Enable estimator for high speeds
Enable_L	Enable estimator for low speeds

Mask Parameters		
Name	**ID**	**Description**
thresh_up	1	Speed threshold for switching from low to high speed estimated values
thresh_down	2	Speed threshold for switching from high to low speed estimated values
n_max	3	Maximum (mechanical) speed (Only used for scaling in fixed point implementations)
ts_fact	4	Multiplication factor of base sampling time (in integer format)

Description:

Implements switching policy between sensorless algorithms for low and high speeds. Decision base is the speed magnitude.
Handle with caution: it is implemented as a switch, so there is no soft transition.

Implementations:

FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
phi_H	int16
phi_L	int16
n_H	int16
n_L	int16

Outports Data Type	
phi	int16
n	int16
Enable_H	bool
Enable_L	bool

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
phi_H	int32
phi_L	int32
n_H	int32
n_L	int32

Outports Data Type	
phi	int32
n	int32
Enable_H	bool
Enable_L	bool

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
phi_H	float32
phi_L	float32
n_H	float32
n_L	float32

Outports Data Type	
phi	float32
n	float32
Enable_H	bool
Enable_L	bool

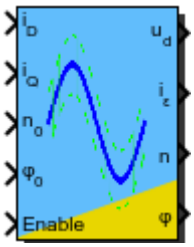
Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
phi_H	float64
phi_L	float64
n_H	float64
n_L	float64

Outports Data Type	
phi	float64
n	float64
Enable_H	bool
Enable_L	bool

Block: HFInjectionSquare



Inports	
iD	Direct current in fixed coordinate system
iQ	Quadrature current in fixed coordinate system
n0	Initial speed preloaded at rising edge of Enable signal
phi0	Initial rotor angle preloaded at rising edge of Enable signal
Enable	Enable == 0: Deactivation of this block; Outputs are set to zero Enable 0->1: Preload of angle and speed for internal integrators Enable == 1: Activation of this block

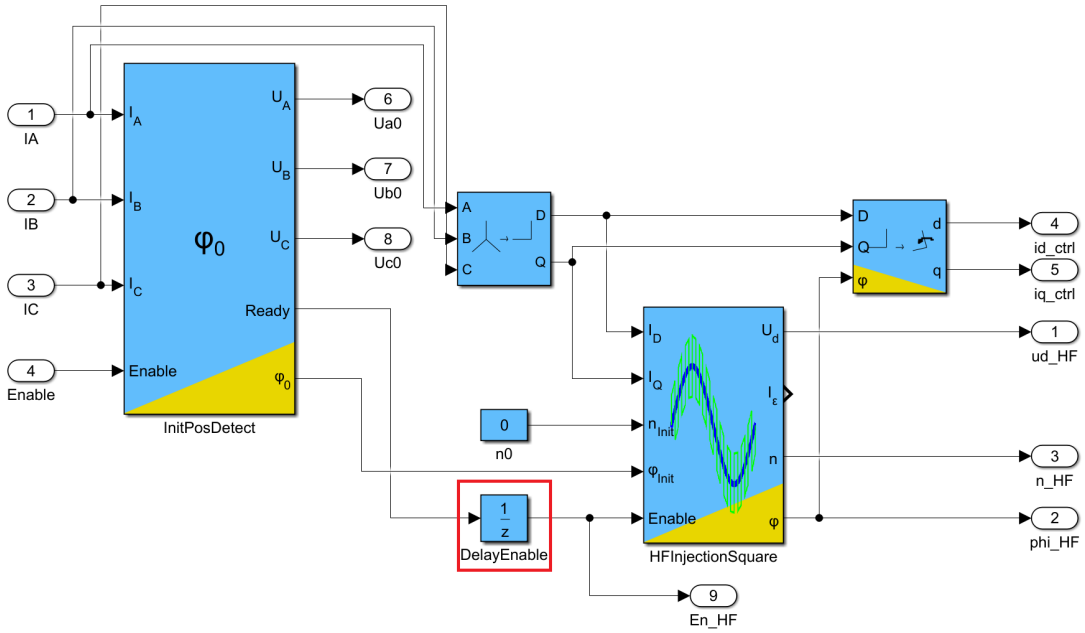
Outports	
ud	Once the HFI is activated, on Ud the square-wave voltage signal is present that is supposed to be injected
i_eps	Error current in q-direction due to HF-injection. (Only needed for parametrization of block.)
n	Angular speed
phi	Estimated electrical rotor angle

Mask Parameters		
Name	**ID**	**Description**
U_inj	1	Amplitude of injected square wave voltage
Jp	2	Moment of inertia of rotor (and load)
I_inj	3	Error current due to HF voltage injection into wrong direction
fo	4	Frequency of observer pole. The higher the pole is chosen, the more dynamic is the angle estimation
p	5	Number of pole pairs
n_max	6	Maximum (mechanical) speed (Only used for scaling in fixed point implementations)
I_max	7	Maximum current (Only used for scaling in fixed point implementations)
U_max	8	Maximum voltage (Only used for scaling in fixed point implementations)
ts_fact	9	Multiplication factor of base sampling time (in integer format)
estimation	10	Select where to place the emphasis of the speed output: dynamic or noise
method	11	Discretization method of of used LTI systems (PI, I)

Description:

High frequency square wave voltage injection:
- Voltage Injection: injects a square-wave voltage signal with fPWM/2 and adjustable amplitude
- Estimator: Calculates rotor angle and speed based on current response due to voltage injection
BE AWARE:
This block needs to be in synchronization with the current controller!

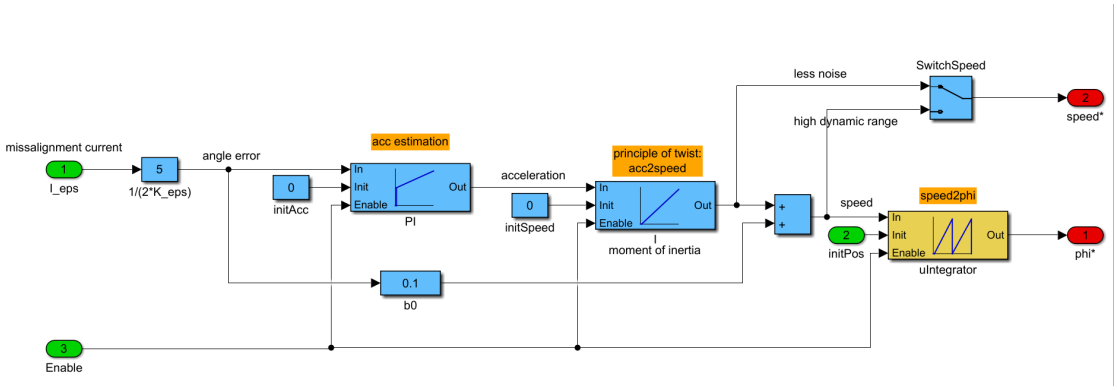
For proper synchronisation with X2C timings, a *Delay*-Block with SampleTimeMultiplier set to 4, must be placed infront of the "Enable" input of the block!



Procedure:
The algorithm mainly spilts into two parts:

1. HF-Injection:
   This part is based on the paper *"Position sensorless control of PMSM by synchronous injection and demodulation of alternating carrier voltage"* by W. Hammel and R. Kennel from 2010.
   A high frequent voltage is injected into the estimated flux direction d. The injected voltage has a constant amplitude but changes its sign with every PWM cycle. Every 4th PWM cycle a misalignment current I_{eps} due to the injected voltage is calculated.
   The aim of the injection is getting the misalignment information when commutating using the estimated rotor angle.

2. Tracking Loop:
   The tracking loop calculates the rotor angle.
   Aim of the tracking loop is to minimize the misalignment current I_{eps} with high dynamics. That way a valid estimate of the rotor angle is obtained.
   The block diagram of the tracking loop is shown in the figure below.
   The parameter of the tracking loop are chosen in a way so that the triple pole is real and negative. The step response of the loop is the same as that of three 1st order delay elements in series.
   The loops initial angle-value has to be provided at the block input φ_{init}. If the starting position is unknown, using the proper parameterized block *InitPosDetect* may be best.
   The interconnection of the blocks is shown in the figure above.



Requirement: The motor must be salient!

Implementations:

FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
iD	int16
iQ	int16
n0	int16
phi0	int16
Enable	bool

Outports Data Type	
ud	int16
i_eps	int16
n	int16
phi	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
iD	int32
iQ	int32
n0	int32
phi0	int32
Enable	bool

Outports Data Type	
ud	int32
i_eps	int32
n	int32
phi	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
iD	float32
iQ	float32
n0	float32
phi0	float32
Enable	bool

Outports Data Type	
ud	float32
i_eps	float32
n	float32
phi	float32

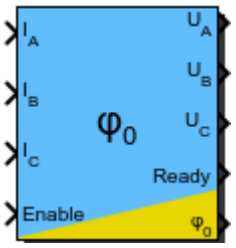
Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
iD	float64
iQ	float64
n0	float64
phi0	float64
Enable	bool

Outports Data Type	
ud	float64
i_eps	float64
n	float64
phi	float64

Block: InitPosDetect



Inports	
I_A	Current in phase A
I_B	Current in phase B
I_C	Current in phase C
Enable	Enable = false: Outputs are zero. Enable = true: Initial rotor angle is determined and enable signal for HF injection is given

Outports	
U_A	Voltage to be applied in phase A
U_B	Voltage to be applied in phase B
U_C	Voltage to be applied in phase C
Ready	Signal to indicate end of initial position detection
phi0	Initial position estimate

Mask Parameters		
Name	**ID**	**Description**
U_pulse	1	Amplitude of voltage pulses used for determination of initial rotor anlge
T_pulse	2	Pulse length in samples
T_pause	3	Time between voltage pulses
U_max	4	Phase voltage scaling value (Only used for scaling in fixed point implementations)
ts_fact	5	Multiplication factor of base sampling time (in integer format)

Description:

Calculation of initial rotor angle.

Procedure:
The algorithm is based on the paper *"Initial rotor angle detection of a nonsalient pole permanent magnet synchronous machine"* by P.B. Schmidt, M. Gasperi, G. Ray, Gorby and A.H. Wijenayake from 1997.

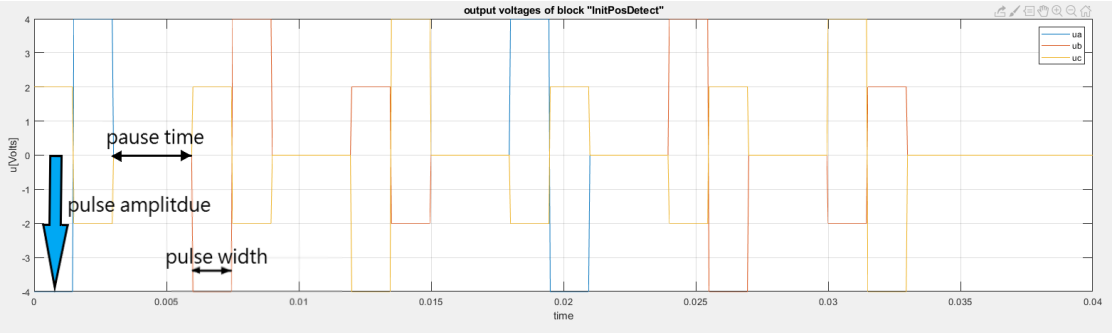


Figure 1: Voltage pulses over time. A counter pulse is also applied!

Above figure shows the output voltages over time.
Six voltage pulses are applied to the motor, not counting the counter pulses. First, a negative voltage pulse with the given amplitude is applied to phase A. The negative half of phase A voltage is applied to phase B and C. After the given pulse time parameter, the counter pulses are applied to the phases.
Second, a negative voltage pulse with the given amplitude is applied to phase B. The negative half of phase B voltage is applied to phase A and C. The pattern is repeated in the same way 6 times in total.

The resulting currents are processed accordingly to calculated the initial rotor angle.

Requirement: To get a proper starting angle, the currents due to the voltage pulses must be large enough to cause material saturation.

Implementations:

FiP16	16 Bit Fixed Point Implementation
FiP32	32 Bit Fixed Point Implementation
Float32	32 Bit Floating Point Implementation
Float64	64 Bit Floating Point Implementation

Implementation: FiP16

16 Bit Fixed Point Implementation

Inports Data Type	
I_A	int16
I_B	int16
I_C	int16
Enable	bool

Outports Data Type	
U_A	int16
U_B	int16
U_C	int16
Ready	bool
phi0	int16

Implementation: FiP32

32 Bit Fixed Point Implementation

Inports Data Type	
I_A	int32
I_B	int32
I_C	int32
Enable	bool

Outports Data Type	
U_A	int32
U_B	int32
U_C	int32
Ready	bool
phi0	int32

Implementation: Float32

32 Bit Floating Point Implementation

Inports Data Type	
I_A	float32
I_B	float32
I_C	float32
Enable	bool

Outports Data Type	
U_A	float32
U_B	float32
U_C	float32
Ready	bool
phi0	float32

Implementation: Float64

64 Bit Floating Point Implementation

Inports Data Type	
I_A	float64
I_B	float64
I_C	float64
Enable	bool

Outports Data Type	
U_A	float64
U_B	float64
U_C	float64
Ready	float64
phi0	float64