ACI_FE	*Flux estimator of the 3-ph induction motor*

Description

This software module implements the flux estimator with the rotor flux angle for the 3-ph induction motor based upon the integral of back emf's (voltage model) approach. To reduce the errors due to pure integrator and stator resistance measurement, the compensated voltages produced by PI compensators are introduced. Therefore, this flux estimator can be operating over a wide range of speed, even at very low speed.

Availability

This IQ module is available in one interface format:

1) The C interface version

Module Properties

Type: Target Independent, Application Dependent

Target Devices: x281x or x280x

C Version File Names: aci_fe.c, aci_fe.h

IQmath library files for C: IQmathLib.h, IQmath.lib

Item	C version	Comments
Code Size□ (x281x/x280x)	380/380 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" ACIFE structure consumes 62 words in the data memory

□ Code size mentioned here is the size of the ***calc()*** function

C Interface

Object Definition

The structure of ACIFE object is defined by following structure definition

```
typedef struct { _iq  ThetaFlux;        // Output: Rotor flux angle
                 _iq  IQsS;             // Input: Stationary q-axis stator current
                 _iq  IDsS;             // Input: Stationary d-axis stator current
                 _iq  K1;               // Parameter: Constant using in current model
                 _iq  FluxDrE;          // Variable: Rotating d-axis rotor flux (current model)
                 _iq  K2;               // Parameter: Constant using in current model
                 _iq  FluxQrS;          // Variable: Stationary q-axis rotor flux (current model)
                 _iq  FluxDrS;          // Variable: Stationary d-axis rotor flux (current model)
                 _iq  K3;               // Parameter: Constant using in stator flux computation
                 _iq  K4;               // Parameter: Constant using in stator flux computation
                 _iq  FluxDsS;          // Variable: Stationary d-axis stator flux (current model)
                 _iq  FluxQsS;          // Variable: Stationary q-axis stator flux (current model)
                 _iq  PsiQsS;           // Variable: Stationary d-axis stator flux (voltage model)
                 _iq  Kp;               // Parameter: PI proportional gain
                 _iq  UiDsS;            // Variable: Stationary d-axis integral term
                 _iq  UCompDsS;         // Variable: Stationary d-axis compensated voltage
                 _iq  Ki;               // Parameter: PI integral gain
                 _iq  PsiQsS;           // Variable: Stationary q-axis stator flux (voltage model)
                 _iq  UiQsS;            // Variable: Stationary q-axis integral term
                 _iq  UCompQsS;         // Variable: Stationary q-axis compensated voltage
                 _iq  EmfDsS;           // Variable: Stationary d-axis back emf
                 _iq  UDsS;             // Input: Stationary d-axis stator voltage
                 _iq  K5;               // Parameter: Constant using in back emf computation
                 _iq  K6;               // Parameter: Constant using in back emf computation
                 _iq  EmfQsS;           // Variable: Stationary q-axis back emf
                 _iq  UQsS;             // Input: Stationary q-axis stator voltage
                 _iq  K8;               // Parameter: Constant using in rotor flux computation
                 _iq  K7;               // Parameter: Constant using in rotor flux computation
                 _iq  PsiDrS;           // Output: Stationary d-axis estimated rotor flux
                 _iq  PsiQrS;           // Output: Stationary q-axis estimated rotor flux
                  void  (*calc)();      // Pointer to calculation function
               } ACIFE;


typedef ACIFE *ACIFE_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format*	Range(Hex)
Inputs	UDsS	stationary d-axis stator voltage	GLOBAL_Q	80000000-7FFFFFFF
	UQsS	stationary q-axis stator voltage	GLOBAL_Q	80000000-7FFFFFFF
	IDsS	stationary d-axis stator current	GLOBAL_Q	80000000-7FFFFFFF
	IQsS	stationary q-axis stator current	GLOBAL_Q	80000000-7FFFFFFF
Outputs	PsiDrS	stationary d-axis rotor flux linkage	GLOBAL_Q	80000000-7FFFFFFF
	PsiQrS	stationary q-axis rotor flux linkage	GLOBAL_Q	80000000-7FFFFFFF
	ThetaFlux	rotor flux linkage angle	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
ACI_FE parameter	K1	K1 = Tr/(Tr+T)	GLOBAL_Q	80000000-7FFFFFFF
	K2	K2 = T/(Tr+T)	GLOBAL_Q	80000000-7FFFFFFF
	K3	K3 = Lm/Lr	GLOBAL_Q	80000000-7FFFFFFF
	K4	K4 = (Ls*Lr-Lm*Lm)/(Lr*Lm)	GLOBAL_Q	80000000-7FFFFFFF
	K5	K5 = Rs*Ib/Vb	GLOBAL_Q	80000000-7FFFFFFF
	K6	K6 = T*Vb/(Lm*Ib)	GLOBAL_Q	80000000-7FFFFFFF
	K7	K7 = Lr/Lm	GLOBAL_Q	80000000-7FFFFFFF
	K8	K8 = (Ls*Lr-Lm*Lm)/(Lm*Lm)	GLOBAL_Q	80000000-7FFFFFFF
Internal	FluxDrE	stationary d-axis rotor flux	GLOBAL_Q	80000000-7FFFFFFF
	FluxQrE	stationary q-axis rotor flux	GLOBAL_Q	80000000-7FFFFFFF
	FluxDsS	stationary d-axis stator flux	GLOBAL_Q	80000000-7FFFFFFF
	FluxQsS	stationary q-axis stator flux	GLOBAL_Q	80000000-7FFFFFFF
	PsiQsS	stationary d-axis stator flux	GLOBAL_Q	80000000-7FFFFFFF
	PsiQsS	stationary q-axis stator flux	GLOBAL_Q	80000000-7FFFFFFF
	UiDsS	stationary d-axis integral term	GLOBAL_Q	80000000-7FFFFFFF
	UiQsS	stationary q-axis integral term	GLOBAL_Q	80000000-7FFFFFFF
	EmfDsS	stationary d-axis back emf	GLOBAL_Q	80000000-7FFFFFFF
	EmfQsS	stationary q-axis back emf	GLOBAL_Q	80000000-7FFFFFFF
	UCompDsS	stationary d-axis comp. voltage	GLOBAL_Q	80000000-7FFFFFFF
	UCompQsS	stationary q-axis comp. voltage	GLOBAL_Q	80000000-7FFFFFFF

* GLOBAL_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

Special Constants and Data types

ACIFE
The module definition is created as a data type. This makes it convenient to instance an interface to the flux estimator of Induction Motor module. To create multiple instances of the module simply declare variables of type ACIFE.

ACIFE_handle
User defined Data type of pointer to ACIFE module

ACIFE_DEFAULTS
Structure symbolic constant to initialize ACIFE module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void aci_fe_calc(ACIFE_handle);

This definition implements one method viz., the flux estimator of Induction Motor computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two ACIFE objects
ACIFE fe1, fe2;

Initialization
To Instance pre-initialized objects
ACIFE fe1 = ACIFE_DEFAULTS;
ACIFE fe2 = ACIFE_DEFAULTS;

Invoking the computation function
fe1.calc(&fe1);
fe2.calc(&fe2);

Example

The following pseudo code provides the information about the module usage.

```
main()
{
        fe1.K1 = parem1_1;              // Pass parameters to fe1
                .                                .
                .                                .
                .                                .
        fe1.K8 = parem1_8;              // Pass parameters to fe1

        fe2.K1 = parem2_1;              // Pass parameters to fe2
                .                                .
                .                                .
                .                                .
        fe2.K10_fe = parem2_8;          // Pass parameters to fe2

}

void interrupt periodic_interrupt_isr()
{
        fe1.UDsS= voltage_dq1.d;        // Pass inputs to fe1
        fe1.UQsS= voltage_dq1.q;        // Pass inputs to fe1
        fe1.IQsS=current_dq1.d;         // Pass inputs to fe1
        fe1.IDsS=current_dq1.q;         // Pass inputs to fe1

        fe2.UDsS= voltage_dq2.d;        // Pass inputs to fe2
        fe2.UQsS= voltage_dq2.q;        // Pass inputs to fe2
        fe2.IQsS=current_dq2.d;         // Pass inputs to fe2
        fe2.IDsS=current_dq2.q;         // Pass inputs to fe2
```

```
        fe1.calc(&fe1);                 // Call compute function for fe1
        fe2.calc(&fe2);                 // Call compute function for fe2

        flux1.d = fe1.PsiDrS;           // Access the outputs of fe1
        flux1.q = fe1.PsiQrS;           // Access the outputs of fe1
       angle1 = fe1.ThetaFlux;          // Access the outputs of fe1

        flux2.d = fe2.PsiDrS;           // Access the outputs of fe2
        flux2.q = fe2.PsiQrS;           // Access the outputs of fe2
        angle2 = fe2.ThetaFlux;         // Access the outputs of fe2

}
```

Constant Computation Function

Since the flux estimator of Induction motor module requires eight constants (K1,…, K8) to be input basing on the machine parameters, base quantities, mechanical parameters, and sampling period. These eight constants can be internally computed by the C function (aci_fe_const.c, aci_fe_const.h). The followings show how to use the C constant computation function.

Object Definition

The structure of ACIFE_CONST object is defined by following structure definition

```
typedef struct { float32  Rs;          // Input: Stator resistance (ohm)
                 float32  Rr;          // Input: Rotor resistance (ohm)
                 float32  Ls;          // Input: Stator inductance (H)
                 float32 Lr;           // Input: Rotor inductance (H)
                 float32  Lm;          // Input: Magnetizing inductance (H)
                 float32  Ib;          // Input: Base phase current (amp)
                 float32  Vb;          // Input: Base phase voltage (volt)
                 float32  Ts;          // Input: Sampling period in sec
                 float32  K1;          // Output: constant using in rotor flux calculation
                 float32  K2;          // Output: constant using in rotor flux calculation
                 float32  K3;          // Output: constant using in rotor flux calculation
                 float32  K4;          // Output: constant using in stator current calculation
                 float32  K5;          // Output: constant using in stator current calculation
                 float32  K6;          // Output: constant using in stator current calculation
                 float32  K7;          // Output: constant using in stator current calculation
                 float32  K8;          // Output: constant using in torque calculation
                 void   (*calc)();     // Pointer to calculation function
               } ACIFE_CONST;

typedef ACIFE_CONST *ACIFE_CONST_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format	Range(Hex)
Inputs	Rs	Stator resistance (ohm)	Floating	N/A
	Rr	Rotor resistance (ohm)	Floating	N/A
	Ls	Stator inductance (H)	Floating	N/A
	Lr	Rotor inductance (H)	Floating	N/A
	Lm	Magnetizing inductance (H)	Floating	N/A
	Ib	Base phase current (amp)	Floating	N/A
	Vb	Base phase voltage (volt)	Floating	N/A
	Ts	Sampling period (sec)	Floating	N/A
Outputs	K1	constant using in rotor flux calculation	Floating	N/A
	K2	constant using in rotor flux calculation	Floating	N/A
	K3	constant using in rotor flux calculation	Floating	N/A
	K4	constant using in stator current cal.	Floating	N/A
	K5	constant using in stator current cal.	Floating	N/A
	K6	constant using in stator current cal.	Floating	N/A
	K7	constant using in stator current cal.	Floating	N/A
	K8	constant using in torque calculation	Floating	N/A

Special Constants and Data types

ACIFE_CONST
The module definition is created as a data type. This makes it convenient to instance an interface to the flux estimation of Induction Motor constant computation module. To create multiple instances of the module simply declare variables of type ACIFE_CONST.

ACIFE_CONST_handle
User defined Data type of pointer to ACIFE_CONST module

ACIFE_CONST_DEFAULTS
Structure symbolic constant to initialize ACIFE_CONST module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void aci_fe_const_calc(ACIFE_CONST_handle);

This definition implements one method viz., the flux estimator of Induction Motor constant computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two ACIFE_CONST objects
ACIFE_CONST fe1_const, fe2_const;

Initialization
To Instance pre-initialized objects
ACIFE_CONST fe1_const = ACIFE_CONST_DEFAULTS;
ACIFE_CONST fe2_const = ACIFE_CONST_DEFAULTS;

Invoking the computation function
fe1_const.calc(&fe1_const);
fe2_const.calc(&fe2_const);

Example

The following pseudo code provides the information about the module usage.

```
main()
{

        fe1_const.Rs = Rs1;             // Pass floating-point inputs to fe1_const
        fe1_const.Rr = Rr1;             // Pass floating-point inputs to fe1_const
        fe1_const.Ls = Ls1;             // Pass floating-point inputs to fe1_const
        fe1_const.Lr = Lr1;             // Pass floating-point inputs to fe1_const
        fe1_const.Lm = Lm1;             // Pass floating-point inputs to fe1_const
        fe1_const.Ib = Ib1;             // Pass floating-point inputs to fe1_const
        fe1_const.Vb = Vb1;             // Pass floating-point inputs to fe1_const
        fe1_const.Ts = Ts1;             // Pass floating-point inputs to fe1_const

        fe2_const.Rs = Rs2;             // Pass floating-point inputs to fe2_const
        fe2_const.Rr = Rr2;             // Pass floating-point inputs to fe2_const
        fe2_const.Ls = Ls2;             // Pass floating-point inputs to fe2_const
        fe2_const.Lr = Lr2;             // Pass floating-point inputs to fe2_const
        fe2_const.Lm = Lm2;             // Pass floating-point inputs to fe2_const
        fe2_const.Ib = Ib2;             // Pass floating-point inputs to fe2_const
        fe2_const.Vb = Vb2;             // Pass floating-point inputs to fe2_const
        fe2_const.Ts = Ts2;             // Pass floating-point inputs to fe2_const

        fe1_const.calc(&fe1_const);     // Call compute function for fe1_const
        fe2_const.calc(&fe2_const);     // Call compute function for fe2_const

        fe1.K1 = _IQ(fe1_const.K1);     // Access the floating-point outputs of fe1_const
                .                                       .
                .                                       .
                .                                       .
        fe1.K8 = _IQ(fe1_const.K8); // Access the floating-point outputs of fe1_const


        fe2.K1 = _IQ(fe2_const.K1);     // Access the floating-point outputs of fe2_const
                .                                       .
                .                                       .
                .                                       .
        fe2.K8 = _IQ(fe2_const.K8);   // Access the floating-point outputs of fe2_const

}
```

Technical Background

The overall of the flux estimator [1] can be shown in Figure 1. The rotor flux linkages in the stationary reference frame are mainly computed by means of the integral of back emf's in the voltage model. By introducing the compensated voltages generated by PI compensators, the errors associated with pure integrator and stator resistance measurement can be taken care. The equations derived for this flux estimator are summarized as follows:

Continuous time:

Firstly, the rotor flux linkage dynamics in synchronously rotating reference frame $\left(\omega = \omega_e = \omega_{\psi_r}\right)$ can be shown as below:

$$\frac{d\psi_{dr}^{e,i}}{dt} = \frac{L_m}{\tau_r} i_{ds}^{e} - \frac{1}{\tau_r}\psi_{dr}^{e,i} + \left(\omega_e - \omega_r\right)\psi_{qr}^{e,i} \tag{1}$$

$$\frac{d\psi_{qr}^{e,i}}{dt} = \frac{L_m}{\tau_r} i_{qs}^{e} - \frac{1}{\tau_r}\psi_{qr}^{e,i} - \left(\omega_e - \omega_r\right)\psi_{dr}^{e,i} \tag{2}$$

where L_m is the magnetizing inductance (H), $\tau_r = \frac{L_r}{R_r}$ is the rotor time constant (sec), and ω_r is the electrically angular velocity of rotor (rad/sec).

In the current model, total rotor flux linkage is aligned into the d-axis component, which is modeled by the stator currents, thus

$$\psi_r^{e,i} = \psi_{dr}^{e,i} \text{ and } \psi_{qr}^{e,i} = 0 \tag{3}$$

Substituting $\psi_{qr}^{e,i} = 0$ into (1)-(2), yields the oriented rotor flux dynamics are

$$\frac{d\psi_{dr}^{e,i}}{dt} = \frac{L_m}{\tau_r} i_{ds}^{e} - \frac{1}{\tau_r}\psi_{dr}^{e,i} \tag{4}$$

$$\psi_{qr}^{e,i} = 0 \tag{5}$$

Note that (4) and (5) are the classical rotor flux vector control equations. Then, the rotor flux linkages in (4)-(5) are transformed into the stationary reference frame performed by inverse park transformation.

$$\psi_{dr}^{s,i} = \psi_{dr}^{e,i}\cos\left(\theta_{\psi_r}\right) - \psi_{qr}^{e,i}\sin\left(\theta_{\psi_r}\right) = \psi_{dr}^{e,i}\cos\left(\theta_{\psi_r}\right) \tag{6}$$

$$\psi_{qr}^{s,i} = \psi_{dr}^{e,i}\sin\left(\theta_{\psi_r}\right) + \psi_{qr}^{e,i}\cos\left(\theta_{\psi_r}\right) = \psi_{dr}^{e,i}\sin\left(\theta_{\psi_r}\right) \tag{7}$$

where θ_{ψ_r} is the rotor flux angle (rad).

Then, the stator flux linkages in stationary reference frame are computed from the rotor flux linkages in (6)-(7)

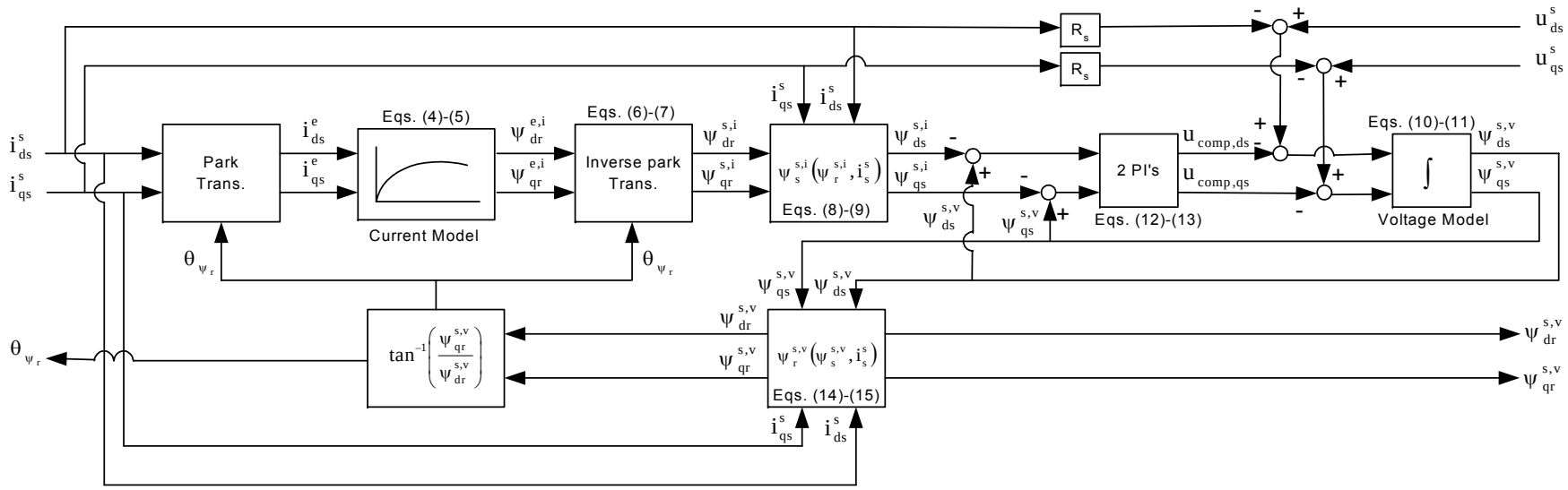

Figure 1: Overall system of flux estimator

$$\psi_{ds}^{s,i} = L_s i_{ds}^s + L_m i_{dr}^s = \left(\frac{L_s L_r - L_m^2}{L_r} \right) i_{ds}^s + \frac{L_m}{L_r} \psi_{dr}^{s,i} \quad (8)$$

$$\psi_{qs}^{s,i} = L_s i_{qs}^s + L_m i_{qr}^s = \left(\frac{L_s L_r - L_m^2}{L_r} \right) i_{qs}^s + \frac{L_m}{L_r} \psi_{qr}^{s,i} \quad (9)$$

where L_s and L_r are the stator and rotor self inductance (H), respectively.

Next, the stator flux linkages in the voltage model is computed by means of back emf's integration with compensated voltages.

$$\psi_{ds}^{s,v} = \int \left(u_{ds}^s - i_{ds}^s R_s - u_{comp,ds} \right) dt \quad (10)$$

$$\psi_{qs}^{s,v} = \int \left(u_{qs}^s - i_{qs}^s R_s - u_{comp,qs} \right) dt \quad (11)$$

where R_s is the stator resistance (Ω), u_{ds}^s, u_{qs}^s are stationary dq-axis stator voltages, and the compensated voltages are computed by the PI control law as follows:

$$u_{comp,ds} = K_p \left(\psi_{ds}^{s,v} - \psi_{ds}^{s,i} \right) + \frac{K_p}{T_I} \int \left(\psi_{ds}^{s,v} - \psi_{ds}^{s,i} \right) dt \quad (12)$$

$$u_{comp,qs} = K_p \left(\psi_{qs}^{s,v} - \psi_{qs}^{s,i} \right) + \frac{K_p}{T_I} \int \left(\psi_{qs}^{s,v} - \psi_{qs}^{s,i} \right) dt \quad (13)$$

The proportional gain K_P and the reset time T_I are chosen such that the flux linkages computed by current model is dominant at low speed because the back emf's computed by the voltage model are extremely low at this speed range (even zero back emf's at zero speed). While at high speed range, the flux linkages computed by voltage model is dominant.

Once the stator flux linkages in (10)-(11) are calculated, the rotor flux linkages based on the voltage model are further computed, by rearranging (8)-(9), as

$$\psi_{dr}^{s,v} = -\left(\frac{L_s L_r - L_m^2}{L_m} \right) i_{ds}^s + \frac{L_r}{L_m} \psi_{ds}^{s,v} \quad (14)$$

$$\psi_{qr}^{s,v} = -\left(\frac{L_s L_r - L_m^2}{L_m} \right) i_{qs}^s + \frac{L_r}{L_m} \psi_{qs}^{s,v} \quad (15)$$

Then, the rotor flux angle based on the voltage model is finally computed as

$$\theta_{\psi_r} = \tan^{-1} \left(\frac{\psi_{qr}^{s,v}}{\psi_{dr}^{s,v}} \right) \quad (16)$$

Discrete time:

The oriented rotor flux dynamics in (4) is discretized by using backward approximation as follows:

$$\frac{\psi_{dr}^{e,i}(k) - \psi_{dr}^{e,i}(k-1)}{T} = \frac{L_m}{\tau_r} i_{ds}^e(k) - \frac{1}{\tau_r} \psi_{dr}^{e,i}(k) \quad (17)$$

where T is the sampling period (sec). Rearranging (17), then it gives

$$\psi_{dr}^{e,i}(k) = \left(\frac{\tau_r}{\tau_r + T} \right) \psi_{dr}^{e,i}(k\text{-}1) + \left(\frac{L_m T}{\tau_r + T} \right) i_{ds}^e(k) \quad (18)$$

Next, the stator flux linkages in (10)-(11) are discretized by using trapezoidal (or tustin) approximation as

$$\psi_{ds}^{s,v}(k) = \psi_{ds}^{s,v}(k-1) + \frac{T}{2}\left(e_{ds}^{s}(k) + e_{ds}^{s}(k-1)\right) \quad (19)$$

$$\psi_{qs}^{s,v}(k) = \psi_{qs}^{s,v}(k-1) + \frac{T}{2}\left(e_{qs}^{s}(k) + e_{qs}^{s}(k-1)\right) \quad (20)$$

where the back emf's are computed as

$$e_{ds}^{s}(k) = u_{ds}^{s}(k) - i_{ds}^{s}(k)R_s - u_{comp,ds}(k) \quad (21)$$

$$e_{qs}^{s}(k) = u_{qs}^{s}(k) - i_{qs}^{s}(k)R_s - u_{comp,qs}(k) \quad (22)$$

Similarly, the PI control laws in (12)-(13) are also discretized by using trapezoidal approximation as

$$u_{comp,ds}(k) = K_p\left(\psi_{ds}^{s,v}(k) - \psi_{ds}^{s,i}(k)\right) + u_{comp,ds,i}(k-1) \quad (23)$$

$$u_{comp,qs}(k) = K_p\left(\psi_{qs}^{s,v}(k) - \psi_{qs}^{s,i}(k)\right) + u_{comp,qs,i}(k-1) \quad (24)$$

where the accumulating integral terms are as

$$\begin{aligned} u_{comp,ds,i}(k) &= u_{comp,ds,i}(k-1) + \frac{K_p T}{T_I}\left(\psi_{ds}^{s,v}(k) - \psi_{ds}^{s,i}(k)\right) \\ &= u_{comp,ds,i}(k-1) + K_p K_I\left(\psi_{ds}^{s,v}(k) - \psi_{ds}^{s,i}(k)\right) \end{aligned} \quad (25)$$

$$\begin{aligned} u_{comp,qs,i}(k) &= u_{comp,qs,i}(k-1) + \frac{K_p T}{T_I}\left(\psi_{qs}^{s,v}(k) - \psi_{qs}^{s,i}(k)\right) \\ &= u_{comp,qs,i}(k-1) + K_p K_I\left(\psi_{qs}^{s,v}(k) - \psi_{qs}^{s,i}(k)\right) \end{aligned} \quad (26)$$

where $K_I = \frac{T}{T_I}$.

Discrete time and Per-unit:

Now all equations are normalized into the per-unit by the specified base quantities. Firstly, the rotor flux linkage in current model (18) is normalized by dividing the base flux linkage as

$$\psi_{dr,pu}^{e,i}(k) = \left(\frac{\tau_r}{\tau_r + T}\right)\psi_{dr,pu}^{e,i}(k-1) + \left(\frac{T}{\tau_r + T}\right)i_{ds,pu}^{e}(k) \quad \text{pu} \quad (27)$$

where $\psi_b = L_m I_b$ is the base flux linkage (volt.sec) and I_b is the base current (amp).

Next, the stator flux linkages in the current model (8)-(9) are similarly normalized by dividing the base flux linkage as

$$\psi_{ds,pu}^{s,i}(k) = \left(\frac{L_s L_r - L_m^2}{L_r L_m}\right)i_{ds,pu}^{s}(k) + \frac{L_m}{L_r}\psi_{dr,pu}^{s,i}(k) \quad \text{pu} \quad (28)$$

$$\psi^{s,i}_{qs,pu}(k) = \left(\frac{L_s L_r - L_m^2}{L_r L_m}\right) i^s_{qs,pu}(k) + \frac{L_m}{L_r}\psi^{s,i}_{qr,pu}(k) \qquad \text{pu} \qquad (29)$$

Then, the back emf's in (21)-(22) are normalized by dividing the base phase voltage V_b

$$e^s_{ds,pu}(k) = u^s_{ds,pu}(k) - \frac{I_b R_s}{V_b} i^s_{ds,pu}(k) - u_{comp,ds,pu}(k) \qquad \text{pu} \qquad (30)$$

$$e^s_{qs,pu}(k) = u^s_{qs,pu}(k) - \frac{I_b R_s}{V_b} i^s_{qs,pu}(k) - u_{comp,qs,pu}(k) \qquad \text{pu} \qquad (31)$$

Next, the stator flux linkages in the voltage model (19)-(20) are divided by the base flux linkage.

$$\psi^{s,v}_{ds,pu}(k) = \psi^{s,v}_{ds,pu}(k-1) + \frac{V_b T}{L_m I_b}\left(\frac{e^s_{ds,pu}(k) + e^s_{ds,pu}(k-1)}{2}\right) \qquad \text{pu} \qquad (32)$$

$$\psi^{s,v}_{qs,pu}(k) = \psi^{s,v}_{qs,pu}(k-1) + \frac{V_b T}{L_m I_b}\left(\frac{e^s_{qs,pu}(k) + e^s_{qs,pu}(k-1)}{2}\right) \qquad \text{pu} \qquad (33)$$

Similar to (28)-(29), the normalized rotor flux linkages in voltage model are

$$\psi^{s,v}_{dr,pu}(k) = -\left(\frac{L_s L_r - L_m^2}{L_m L_m}\right) i^s_{ds,pu}(k) + \frac{L_r}{L_m}\psi^{s,v}_{ds,pu}(k) \qquad \text{pu} \qquad (34)$$

$$\psi^{s,v}_{qr,pu}(k) = -\left(\frac{L_s L_r - L_m^2}{L_m L_m}\right) i^s_{qs,pu}(k) + \frac{L_r}{L_m}\psi^{s,v}_{qs,pu}(k) \qquad \text{pu} \qquad (35)$$

In conclusion, the discrete-time, per-unit equations are rewritten in terms of constants.

Current model – rotor flux linkage in synchronously rotating reference frame $\left(\omega = \omega_{\psi_r}\right)$

$$\psi^{e,i}_{dr,pu}(k) = K_1 \psi^{e,i}_{dr,pu}(k-1) + K_2 i^e_{ds,pu}(k) \qquad \text{pu} \qquad (36)$$

where $K_1 = \dfrac{\tau_r}{\tau_r + T}$, and $K_2 = \dfrac{T}{\tau_r + T}$.

Current model – rotor flux linkages in the stationary reference frame $\left(\omega = 0\right)$

$$\psi^{s,i}_{ds,pu}(k) = K_4 i^s_{ds,pu}(k) + K_3 \psi^{s,i}_{dr,pu}(k) \qquad \text{pu} \qquad (37)$$

$$\psi^{s,i}_{qs,pu}(k) = K_4 i^s_{qs,pu}(k) + K_3 \psi^{s,i}_{qr,pu}(k) \qquad \text{pu} \qquad (38)$$

where $K_3 = \dfrac{L_m}{L_r}$, and $K_4 = \dfrac{L_s L_r - L_m^2}{L_r L_m}$.

Voltage model – back emf's in the stationary reference frame $\left(\omega = 0\right)$

$$e^s_{ds,pu}(k) = u^s_{ds,pu}(k) - K_5 i^s_{ds,pu}(k) - u_{comp,ds,pu}(k) \qquad \text{pu} \qquad (39)$$

$$e^s_{qs,pu}(k) = u^s_{qs,pu}(k) - K_5 i^s_{qs,pu}(k) - u_{comp,qs,pu}(k) \qquad \text{pu} \qquad (40)$$

where $K_5 = \dfrac{I_b R_s}{V_b}$.

Voltage model – stator flux linkages in the stationary reference frame $\left(\omega = 0\right)$

$$\psi^{s,v}_{ds,pu}(k) = \psi^{s,v}_{ds,pu}(k-1) + K_6\left(\frac{e^{s}_{ds,pu}(k) + e^{s}_{ds,pu}(k-1)}{2}\right) \quad \text{pu} \quad (41)$$

$$\psi^{s,v}_{qs,pu}(k) = \psi^{s,v}_{qs,pu}(k-1) + K_6\left(\frac{e^{s}_{qs,pu}(k) + e^{s}_{qs,pu}(k-1)}{2}\right) \quad \text{pu} \quad (42)$$

where $K_6 = \dfrac{V_b T}{L_m I_b}$.

Voltage model – rotor flux linkages in the stationary reference frame $(\omega = 0)$

$$\psi^{s,v}_{dr,pu}(k) = -K_8 i^{s}_{ds,pu}(k) + K_7 \psi^{s,v}_{ds,pu}(k) \quad \text{pu} \quad (43)$$

$$\psi^{s,v}_{qr,pu}(k) = -K_8 i^{s}_{qs,pu}(k) + K_7 \psi^{s,v}_{qs,pu}(k) \quad \text{pu} \quad (44)$$

where $K_7 = \dfrac{L_r}{L_m}$, and $K_8 = \dfrac{L_s L_r - L_m^2}{L_m L_m}$.

Voltage model – rotor flux angle

$$\theta_{\psi_r,pu}(k) = \frac{1}{2\pi}\tan^{-1}\left(\frac{\psi^{s,v}_{qr,pu}(k)}{\psi^{s,v}_{dr,pu}(k)}\right) \quad \text{pu} \quad (45)$$

Notice that the rotor flux angle is computed by a look-up table of 0^o-45^o with 256 entries.

In fact, equations (36)-(44) are mainly employed to compute the estimated flux linkages in per-unit. The required parameters for this module are summarized as follows:

The machine parameters:
- stator resistance (R_s)
- rotor resistance (R_r)
- stator leakage inductance (L_{sl})
- rotor leakage inductance (L_{rl})
- magnetizing inductance (L_m)

The based quantities:
- base current (I_b)
- base phase voltage (V_b)

The sampling period:
- sampling period (T)

Notice that the stator self inductance is $L_s = L_{sl} + L_m$ (H) and the rotor self inductance is $L_r = L_{rl} + L_m$ (H).

Next, Table 1 shows the correspondence of notations between variables used here and variables used in the program (i.e., aci_fe.c, aci_fe.h). The software module requires that both input and output variables are in per unit values.

	Equation Variables	Program Variables
Inputs	u_{ds}^{s}	UDsS
	u_{qs}^{s}	UQsS
	i_{ds}^{s}	IDsS
	i_{qs}^{s}	IQsS
Outputs	$\psi_{dr}^{s,v}$	PsiDrS
	$\psi_{qr}^{s,v}$	PsiQrS
	θ_{ψ_r}	ThetaFlux
Others	$\psi_{dr}^{e,i}$	FluxDrE
	$\psi_{dr}^{s,i}$	FluxDrS
	$\psi_{qr}^{s,i}$	FluxQrS
	$\psi_{ds}^{s,i}$	FluxDsS
	$\psi_{qs}^{s,i}$	FluxQsS
	$\psi_{ds}^{s,v}$	PsiQsS
	$\psi_{qs}^{s,v}$	PsiQsS
	e_{ds}^{s}	EmfDsS
	e_{qs}^{s}	EmfQsS
	$u_{comp,ds}$	UCompDsS
	$u_{comp,qs}$	UCompQsS

Table 1: Correspondence of notations

References:

[1] C. Lascu, I. Boldea, and F. Blaabjerg, “A modified direct torque control for induction motor sensorless drive”, *IEEE Trans. Ind. Appl.*, vol. 36, no. 1, pp. 122-130, January/February 2000.