Technical Reports

1. Parameter Identification of SRM from Standstill Test
2. Sensorless Control of SRM using Sliding Mode Observers

Professor: ***Ali Keyhani***
Ph.D. Students: ***Wenzhe Lu, Min Dai***

Mechatronics Laboratory
Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210

Oct. 20, 2000

Technical Report 1

Parameter Identification of Switched Reluctance Motors from Standstill Test

Contents included:

1. Introduction

This research is a part of the EMB (Electro-Mechanical Brake) project. The goal of the project is to develop and implement a low-cost sensorless SRM (Switched Reluctance Motor) drive system.

During the past months, an inductance-based model of switched reluctance motor has already been successfully setup and simulated in Simulink. Now we need to identify the parameters of a real SRM to validate the model. This is first done with the standstill test. Then, based on the results of standstill test, on-line identification will be performed to improve the estimated parameters.

The phase inductance L of an SRM changes with rotor position θ and phase current i. Obtaining an accurate model of the phase inductance is the key issue for switched reluctance motor modeling.

For the inductance-based SRM model, the nonlinear inductance $L(\theta, i)$ can be approximated by using three quantities L_a, L_m, and L_u, which denote the inductance at aligned position, unaligned position and a midway between the two. To measure the inductances at these three particular rotor positions, it is necessary to determine these three positions precisely and block the rotor exactly at these positions.

For an 8/6 SRM, this is easy. For example, if A, B, C, and D are the four phases of a 8/6 SRM and only phase A is excited, then the rotor will stop at the aligned position for phase A. And coincidently, the rotor is right at the unaligned position for phase C and the midway position for both phase B and D (Figure 1). Therefore, if the rotor is locked in this case, L_a can be test from phase A, L_u can be tested from phase C, and L_m can be tested from either phase B or phase D.

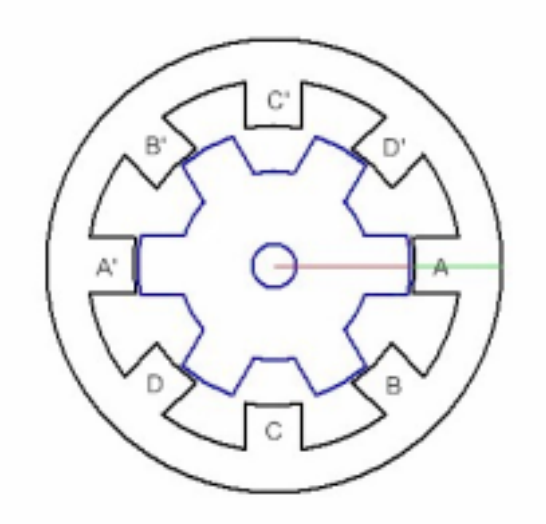


Figure 1. Determining the Rotor Position for L_a, L_m, and L_u Measurement

The basic idea of standstill test is that,

- Move the rotor to a specific position (aligned, unaligned, or a midway between the two) and block it;
- Inject a voltage pulse to a phase winding;
- Measure the current generated in the phase winding;
- Select a model structure for the phase winding;
- Use Maximum Likelihood Estimation (MLE) to identify the parameters (resistances and inductances);
- Validate the model structure.

The test circuit is shown in Figure 2. A power converter is used to provide the excitation of a certain waveform to the winding.

To do this, we need to setup a testbed that can generate desired switching signal for the power converter and can sample the real-time voltage and current in the phase winding. dSPACE DS1103 controller board system and SEMIKRON flexible power converter are used here to perform the task, which is detailed in the following sections.

2. Experimental Setup for Standstill Test

The structure of the SRM Drive hardware is shown in Figure 2.

All signals (analog and digital) between DS1103 controller board and SRM system are connected through an interface box. Four phase voltages (Va, Vb, Vc, and Vd) and four phase currents (Ia, Ib, Ic, and Id) are measured and sent to A/D channels of DS1103 through a signal-conditioning box. Rotor position is sensed by two hall sensors and sent to the capture input ports of slave DSP. The control signal is output from the digital I/O ports of master PPC and sent to two

Semikron power converter boxes to provide appropriate voltages to phase windings of SRM.

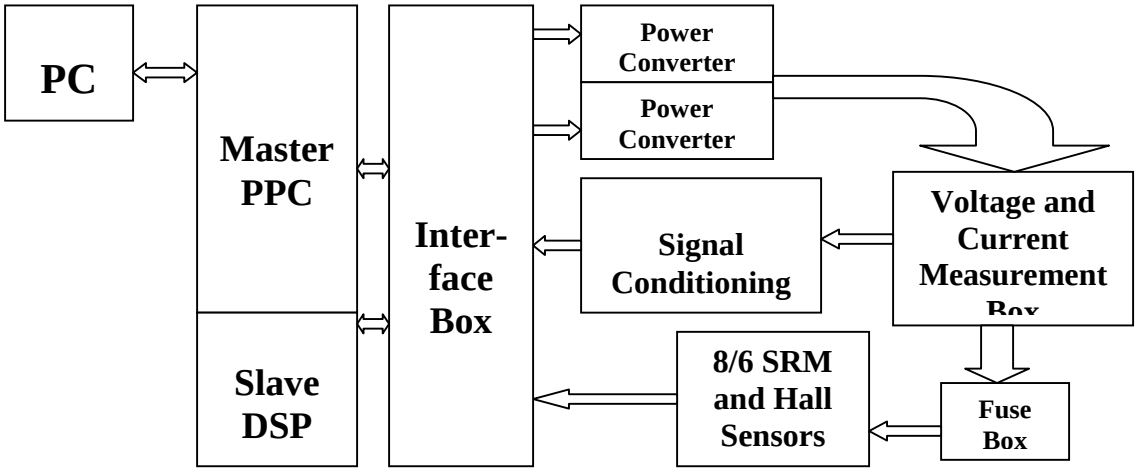


Figure 2. Structure of SRM Drive with DS1103

The Semikron power converter is an integrated 3-phase full-bridge converter system. We use part of the switches to form the power converter for SRM experimental setup. The connection between Semikron power converters and SRM is shown in Figure 3.

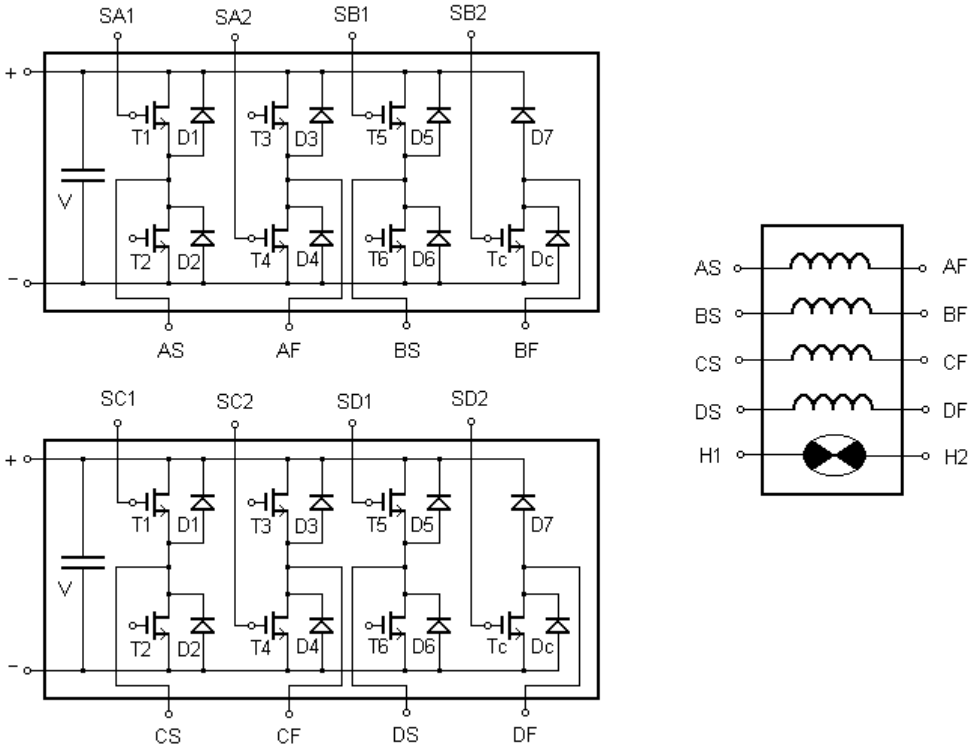


Figure 3. Connection between Semikron power converters and SRM

The control signals (SA1 ~ SD2) are generated through digital I/O ports of master PPC:

SA1	SA2	SB1	SB2	SC1	SC2	SD1	SD2
IO0*	IO1	IO2	IO3	IO4	IO5	IO6	IO7
P2B-18	P2A-18	P2B-02	P2A-02	P2B-19	P2A-19	P2B-03	P2A-03

*: All signal names are referred to those defined in *dSPACE Installation and Configuration Guide.* The third row shows the connector pin number of the corresponding signal.

The phase voltages and currents are sampled by the 16-bit A/D channels of master PPC:

Ia	Va	Ib	Vb	Ic	Vc	Id	Vd
ACH2	ACH4	ACH6	ACH8	ACH10	ACH12	ACH14	ACH16
P1A-34	P1A-02	P1A-19	P1A-36	P1A-04	P1A-21	P1A-38	P1A-06

The rotor position signal is read by the capture input ports of the slave DSP:

H1	H2
SCAP1	SCAP3
P2B-30	P2B-14

There is a fuse box in series with the phase windings. It is used to protect the motor from burning with over current. In the control program, there is also an over-current protection subroutine which turns off the power switches once the current exceeds rated values.

Also, in standstill test, a rheostat is connected in series with the phase winding to prevent the phase current from rising too high.

3. C and Matlab Programs for Data Acquisition

A C program is created for running on the DS1103 Master PowerPC, which performs the following functions:

- Phase voltage and current sampling and A/D conversion;
- Over current protection;
- Rotor position detection;

- Voltage pulse generating and phase current control.

It contains a main program and a timer interrupt subroutine. A/D sampling and conversion, over current protection, and rotor position detection are performed in the interrupt subroutine. The main program performs the initialization, then it waits for operating commands and generates appropriate voltage pulses to corresponding phase windings according to the command. The flowchart of the program is shown in Figure 4.

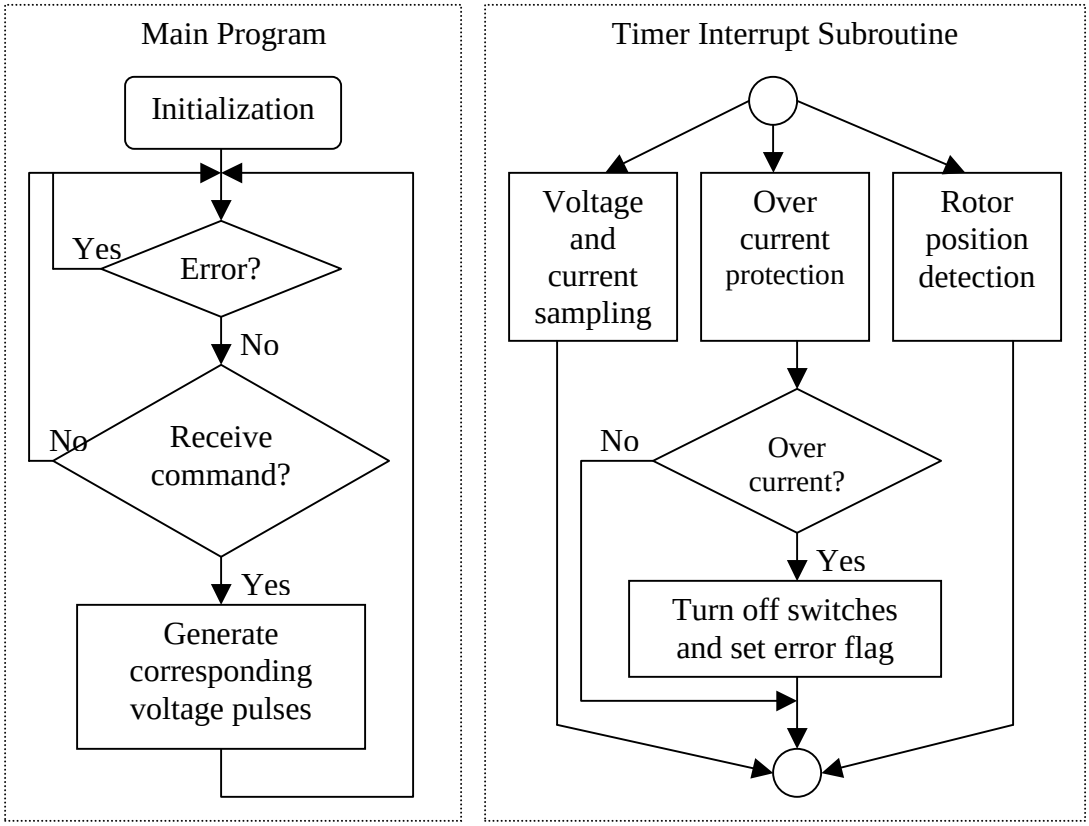


Figure 4. Flowchart of DS1103 Master PPC program

The C code of this program is attached in the appendix.

To start the test and capture the data during standstill test, several Matlab programs are written which make use of the MLIB/MTRACE functions provided by dSPACE. MLIB/MTRACE provides a way for Matlab programs to access real-time variables on dSPACE hardware. So we can send commands to the program running on dSPACE Master PPC and retrieve real-time data from it.

First a Matlab program ‘align (PhaseNo)’ is used to make the rotor move to a position where the rotor is aligned with the stator of the specified phase. When PPC receives this command, it will produce a series of pulses to the phase winding specified by the parameter ‘PhaseNo’. Then the rotor will move to and

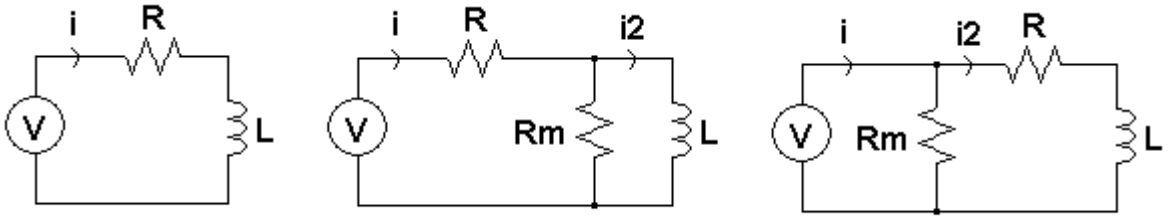
stay at the desired aligned position. During this period, a current control program will be activated to provide hysteresis current to the phase winding.

When the rotor is moved to a desired position, it will be blocked for standstill test. Another Matlab program ‘sst (PhaseNo)’ is used to perform the test on the phase winding specified by ‘PhaseNo’. It first checks whether the real-time program is running on dSPACE, then it sets the pulse parameters such as pulse width and so on. When everything is ready, it will send the command to start the standstill test and capture the real-time data (phase voltage and current). All the captured data will be saved to disk files for later process.

The Matlab programs for ‘align(PhaseNo)’ and ‘sst(PhaseNo)’ are attached in the appendix.

4. Maximum Likelihood Estimation

For a real phase winding, in its parameter identification, it may be modeled multiple possible ways as shown in Figure 5. In Figure 5(a), the phase winding is modeled as an inductor in series of a resistor; in 5(b), a second resistor *Rm* (representing the core losses) may be put in parallel with the inductor; and in 5(c), the second resistor may be in parallel with both the inductor and the resistor. Parameter identification process can be based on any of these structures and the performance can be compared to choose the most appropriate one.



(a) (b) (c)

Figure 5. Three Possible Models for a Phase Winding

Considering the effects of noises caused by the converter harmonics and the measurement, certain identification techniques should be applied to estimate the value of the phase inductance. A maximum likelihood (ML) estimation technique can be used to perform the identification based on dynamic response of the system represented by

$$\begin{cases} X(k+1) = A(\theta_p)X(k) + B(\theta_p)u(k) + w(k) \\ Y(k) = C(\theta_p)X(k) + v(k) \end{cases}, \qquad (1)$$

where θ_p, represents the system parameters, $X(k)$ represents the branch currents (states of the system) in the models shown in Figure 5, $Y(k)$ represents the terminal current that is measurable, $u(k)$ is the excitation voltage, $w(k)$ is the process noise and $v(k)$ is the measurement noise.

The maximum likelihood estimation is performed based on the mechanism shown in Figure 6. In Figure 6, the first block represents the real winding to be estimated. The second block represents the model of the winding and the Kalman filter. The error between the estimated output (from the second block) and the measured output (from the first block) are sent to the third block – the maximum likelihood estimation block. Then the system parameters will be updated based on the estimation error. This process is repeated till the error is small than a desired value.

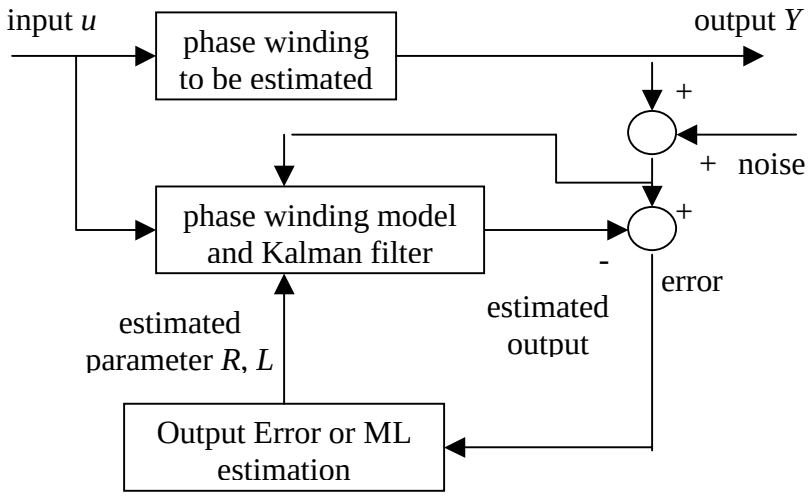


Figure 6. Block diagram of maximum likelihood estimation

In Matlab, the maximum likelihood estimation can be realized by two library functions – leastsq() and kalman(). The first one performs output error estimation when there exist immeasurable states in the system. The second one can build the Kalman filter based on system structure and specified noise covariance. A combination of output error estimator and Kalman filter is nothing but a maximum likelihood estimator.

The Matlab programs for above algorithm are attached in the appendix. The leastsq() function needs an error function that computes the estimation error between estimated output and measurements. The Kalman filter is realized in the error function. It is built based on the estimated system parameters and specified noise covariance. Then it is used to estimate the output based on the input and measurements (with noise). The estimation error can be get by subtract estimated output from measured output.

For different model structures, we only need to modify the part of the program that computes state-space matrices (A, B, C, and D) from system parameters (R, L, and Rm). The other part of program will remain the same.

5. Estimation Results and Analysis

Using the above Matlab program, the data collected from standstill are used to estimate the phase winding parameters. Here is a list of the results. (The tests are repeated several times on each rotor position)

- Aligned position

Table 1: Results for aligned position (model structure [a])

Test No.	Estimated results		Estimation error covariance
	R (ohm)	L (H)	
1	1.052812	0.013227	0.0023
2	1.032301	0.013227	0.0024
3	1.002800	0.013249	0.0022
4	1.015154	0.013362	0.0024
5	1.002230	0.013306	0.0022
6	1.037139	0.013169	0.0024
7	1.043312	0.013273	0.0024
8	1.044221	0.013254	0.0024

The mean value of Ra (aligned position) is 1.028746 ohm, and the covariance of these test results is 0.000382. The mean value of La is 0.013258 H, and the covariance is 0.000000.

Note that the estimation error covariance in the last column of the tables is the covariance of the error between estimated output and measured output when the algorithm converges.

- Unaligned position

Table 2: Results for unaligned position (model structure [a])

Test No.	Estimated results		Estimation error covariance
	R (ohm)	L (H)	
1	0.963861	0.002464	0.0049
2	0.954699	0.002425	0.0049
3	0.964686	0.002420	0.0050
4	0.946639	0.002431	0.0047
5	0.973916	0.002504	0.0047
6	0.954530	0.002435	0.0043
7	0.945639	0.002440	0.0042
8	0.975728	0.002396	0.0046

The mean value of Ru (unaligned position) is 0.959962 ohm, and the covariance of these test results is 0.000132. The mean value of Lu is 0.002439 H, and the covariance is 0.000000.

For 8/6 SRM, when one phase in at aligned position, there are two adjacent phases at their midway position. Standstill tests are performed on both phases.

- Midway position 1

Table 3: Results for midway position 1 (model structure [a])

Test No.	Estimated results		Estimation error covariance
	R (ohm)	L (H)	
1	0.962208	0.007107	0.0029
2	0.970581	0.007062	0.0029
3	0.965988	0.007005	0.0027
4	0.965667	0.007062	0.0027
5	0.985024	0.007060	0.0030
6	0.970776	0.006995	0.0030
7	0.958460	0.007120	0.0032
8	0.959291	0.007132	0.0029

The mean value of Rm1 (midway position 1) is 0.967249 ohm, and the covariance of these test results is 0.000073. The mean value of Lm1 is 0.007068 H, and the covariance is 0.000000.

- Midway position 2

Table 4: Results for midway position 2 (model structure [a])

Test No.	Estimated results		Estimation error covariance
	R (ohm)	L (H)	
1	0.933171	0.007366	0.0030
2	0.959650	0.007275	0.0031
3	0.936856	0.007369	0.0032
4	0.941571	0.007271	0.0029
5	0.957128	0.007212	0.0030
6	0.959995	0.007156	0.0029
7	0.960334	0.007315	0.0029
8	0.938234	0.007327	0.0030

The mean value of Rm2 (midway position 2) is 0.948367 ohm, and the covariance of these test results is 0.000142. The mean value of Lm2 is 0.007286 H, and the covariance is 0.000000.

The resistance of the phase winding is the mean of Ra, Ru, Rm1 and Rm2. The result is R = 0.976081 ohm, and the covariance = 0.001293.

Here is a summary of the standstill test results:

- R = 0.9761 ohm
- La = 13.26 mH
- Lu = 2.44 mH
- Lm = 7.18mH

- Other Model structures

About other model structures, the maximum likelihood algorithm converges to similar R and L values as in model structure (a). And the core loss resistance Rm converges to a large value (greater than 2k ohm) that will is negligible in the circuit. This means model structure (a) is accurate enough for the phase winding model of the tested SR motor.

- Model validation

The estimated parameters are put into the model structure and simulated with the input used for standstill test. The simulated output is compared with the measured output. The error covariance is listed in the above tables. It's obvious that the model and the estimated parameters match the real phase winding very well.

The Matlab program used for model validation is listed in the appendix. Here are some waveforms of the simulation. It's clear that the simulated output matched the measurement very well.

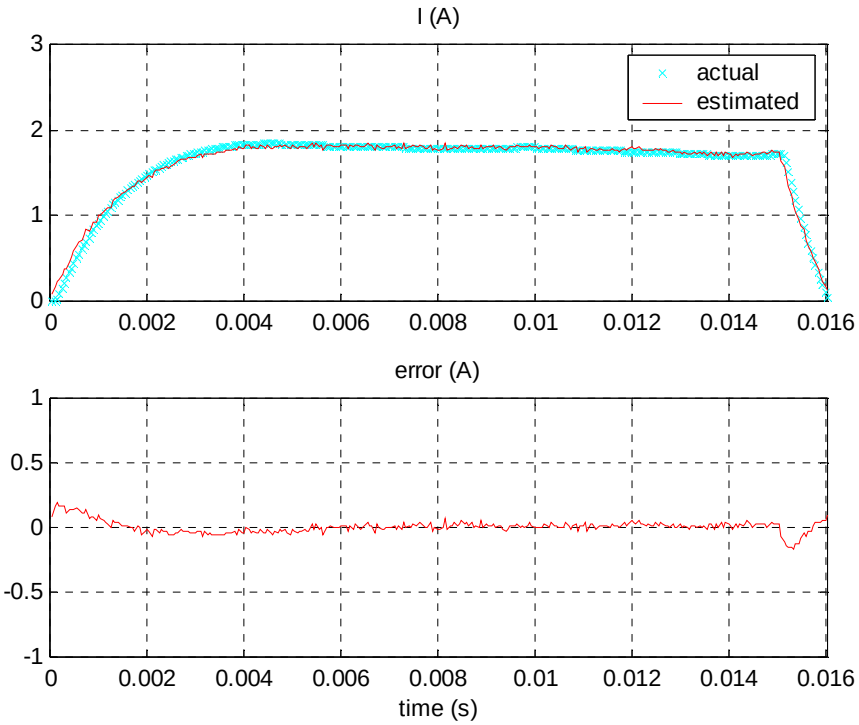


Figure 7. Model validation for aligned position

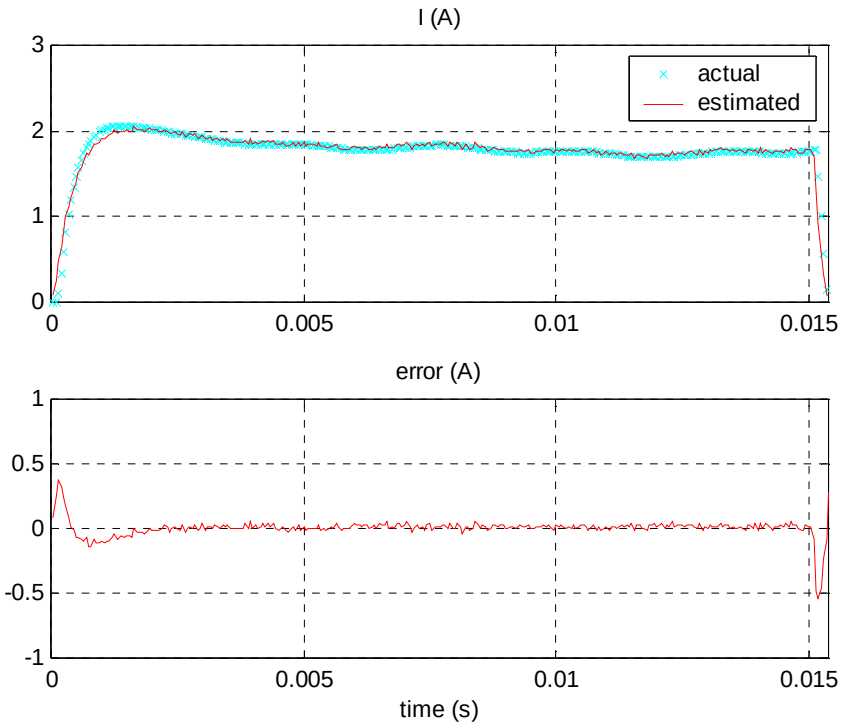


Figure 8. Model validation for unaligned position

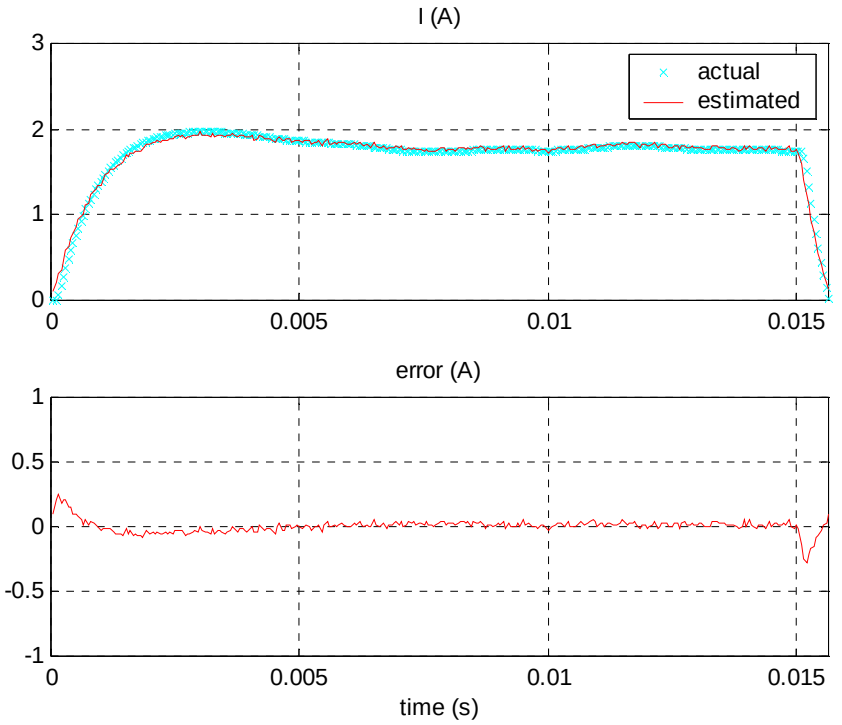


Figure 9. Model validation for midway position

6. Future Work

Since the inductances La and Lm are functions of phase current, they must be updated by online test – to rotate the motor with different load (so phase current is different), record a sequence of data at different rotor positions, and then use maximum likelihood estimation to update the phase winding parameters.

All above tests are performed on the experimental switched reluctance motor. The tests will be repeated on the real SRM from Delphi to get its parameters.

After that, the motor parameters will be put into the EMB model, together with the converter and caliper parameters. And the whole EMB model will be simulated and tested. A controller model will be designed based on the system model. Then rapid control prototyping and hardware-in-the-loop simulation will be performed to test the controller.

7. References

[1] A. Keyhani and S. I, Moon, “Maximum Likelihood Estimation of Synchronous Machine Parameters and Study of Noise Effect From DC Flux Decay Data”, *IEE Proceedings-C*, Vol. 139, No. 1, January 1992.

[2] A. Keyhani, Shangyou Hao, and Richard P. Schulz, “Maximum Likelihood Estimation of Generator Stability Constants Using SSFR Test Data”, *IEEE Transactions on Energy Conversion*, Vol. 6, No. 1, March 1991.

[3] Installation and Configuration Guide, *dSpace Inc*.

[4] DS1103 Feature Reference, *dSpace Inc*.

[5] DS1103 Implementation Guide, *dSpace Inc*.

[6] DS1103 Hardware Reference, *dSpace Inc*.

[7] DS1103 RTI Reference, *dSpace Inc*.

[8] DS1103 RTLib Reference, *dSpace Inc*.

[9] ControlDesk Experiment Guide, *dSpace Inc*.

[10] MATLAB-dSpace Interface Libraries, *dSpace Inc*.

Technical Report 2

Sensorless Control of Switched Reluctance Motors using Sliding Mode Observers

Contents included:

1. Introduction

Rotor position sensing is an integral part of SRM control system due to the torque-production principle of the SRM. Conventionally, a shaft position sensor is employed to detect rotor position. But this means additional cost, more space requirement and an inherent source of unreliability. A sensorless (without direct position sensors) control system, which extracts rotor position information indirectly from electrical or other signals, is expected.

A large amount of sensorless control techniques have been published in the last decade. All these methods have their own advantages and disadvantages. Ideally, it is desirable to have a sensorless scheme, which uses only operating data measured from motor terminals while maintaining a reliable operation over the entire speed and torque range with high resolution and accuracy. Sliding mode observer, with its advantages of inherent robustness of parameters uncertainty, computational simplicity, and high stability, provides a way to implement such 'ideal' sensorless schemes.

2. Review of Inductance Model of SRM

To define a sliding mode observer for SRM drive system, we need first to setup a model of the system and build the system differential equations.

The inductance model of SRM introduced in previous technical report is used here. In this model, "the position dependency of the phase inductance is represented by a limited number of Fourier series terms and the nonlinear variation of the inductance with current is expressed by means of polynomial functions" [2]. The coefficients of the terms in the Fourier series are determined by

three special inductances: L_a (inductance at aligned position), L_u (inductance at unaligned position), and L_m (inductance at a midway between the above two positions), as shown in figure 1.

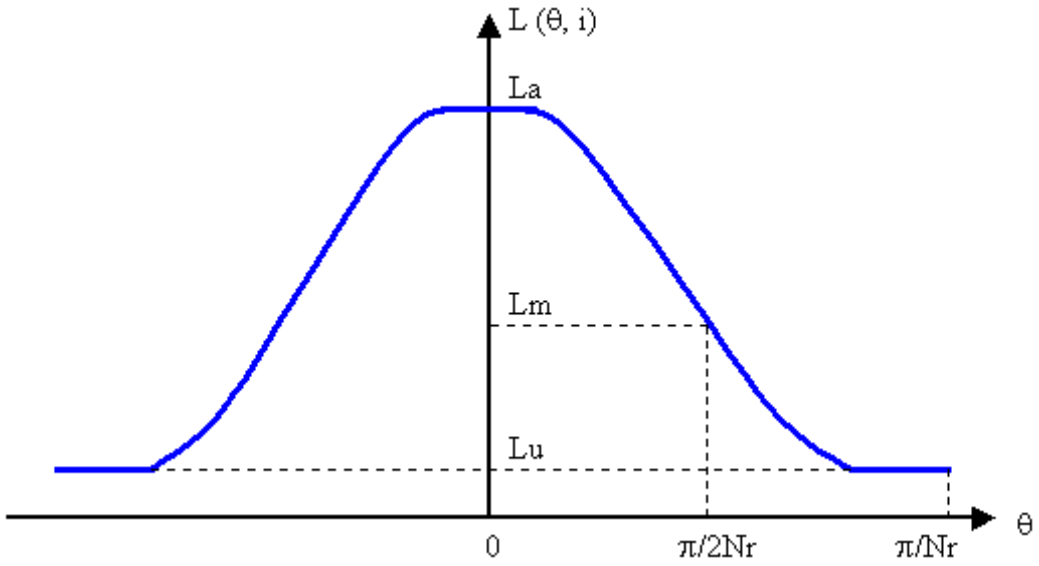


Figure 1. Phase inductance profile of SRM

The phase inductance is expressed as:

$$L(\theta,i) = L_0(i) + L_1(i)\cos N_r\theta + L_2(i)\cos 2N_r\theta\,, \tag{1}$$

where N_r is the number of rotor poles, and

$$L_0 = \frac{1}{2}[\frac{1}{2}(L_a + L_u) + L_m]\,,$$

$$L_1 = \frac{1}{2}(L_a - L_u)\,, \tag{2}$$

$$L_2 = \frac{1}{2}[\frac{1}{2}(L_a + L_u) - L_m]\,.$$

$L_a(i)$ and $L_m(i)$ are approximated by polynomial functions as

$$L_a = \sum_{n=0}^{k} a_n i^n\,, \tag{3}$$

and

$$L_m = \sum_{n=0}^{k} b_n i^n\,. \tag{4}$$

Based on the inductance model, the phase voltage equations can be formed and the phase torque can be computed by the partial derivative of magnetic coenergy with respect to rotor angle θ.

3. System Differential Equations

The SRM drive system differential equations include the electromagnet equations (voltage equations), the electromechanic equations (torque-speed equations), and the mechanic equations, which are detailed below:

Voltage Equations:

The phase voltage equations can be expressed as

$$V_j = R \cdot i_j + \frac{d\lambda_j}{dt} = R \cdot i_j + \frac{d(L_j(\theta, i_j) \cdot i_j)}{dt}. \quad (5)$$

By converting to "standard" differential equations, eq. (5) can be represented as

$$\dot{i}_j = f_1(\theta, i_j, \omega) + g(\theta, i_j, \omega) \cdot V_j, \quad (6)$$

where

$$f_1(\theta, i_j, \omega) = \frac{-R - \frac{\partial L_j}{\partial \theta}\omega}{L_j + i_j \frac{\partial L_j}{\partial i_j}} i_j,$$

and

$$g_1(\theta, i_j, \omega) = \frac{1}{L_j + i_j \frac{\partial L_j}{\partial i_j}}.$$

The voltage equations (j=1…m, where m is the number of phases) are nonlinear differential equations.

Torque-speed Equations:

The torque-speed equation can be expressed as

$$J \frac{d\omega}{dt} = T_e - T_l, \quad (7)$$

where J is the moment of inertia of the rotor, T_l is the load torque, and T_e is the electromagnetic torque. T_e can be computed as follows,

$$T_e = \sum_{j=1}^{m} T_j = \sum_{j=1}^{m} \frac{\partial W_{c,j}}{\partial \theta} = \sum_{j=1}^{m} \frac{\partial \int L_j(\theta, i_j) i_j di}{\partial \theta}.$$

By converting to “standard” differential equations, eq. (7) can be represented as

$$\dot{\omega} = f_2(\theta, i, \omega), \tag{8}$$

where

$$f_2(\theta, i, \omega) = \frac{1}{J}(T_e - T_l).$$

It’s also a nonlinear differential equation.

Mechanic Equations:

The mechanic equations can be expressed as

$$\frac{d\theta}{dt} = \omega, \tag{9}$$

or in “standard” form,

$$\dot{\theta} = \omega, \tag{10}$$

Equations ***(6), (8), and (10)*** form the system differential equations, which are used to define the sliding mode observers.

4. Sliding Mode Observers

A speed and rotor position observer for SRM is based on the following ideas:

- A real motor is running, and the phase voltages and currents are measurable;
- A model of the motor is simulated with the same voltage inputs. The phase currents are estimated;
- The difference between the actual phase currents and the estimated currents are used by the observer to estimate rotor speed and position.

According to the system differential equations derived from the inductance model of SRM, a sliding mode observer for rotor position and speed can be defined as follows,

$$\dot{\hat{i}}_j = f_1(\hat{\theta}, \hat{i}_j, \hat{\omega}) + g(\hat{\theta}, \hat{i}_j, \hat{\omega}) \cdot V_j, \tag{11}$$

$$\dot{\hat{\theta}} = \hat{\omega} + L_1 sign\left(\sum_{j=1}^{m} (i_j - \hat{i}_j)\right), \tag{12}$$

$$\dot{\hat{\omega}} = f_2(\hat{\theta},\hat{i},\hat{\omega}) + L_2 sign(\sum_{j=1}^{m}(i_j - \hat{i}_j)), \quad (13)$$

where $\hat{\theta},\hat{i},\hat{\omega}$ are the estimation of θ,i,ω.

$e_\theta = \theta - \hat{\theta}$, and

$e_\omega = \omega - \hat{\omega}$.

By subtracting eq. (12) from (10), and (13) from (8), we get the error dynamics,

$$\dot{e}_\theta = e_\omega - L_1 sign(\sum_{j=1}^{m}(i_j - \hat{i}_j)) \quad (14)$$

$$\dot{e}_\omega = \Delta f_2 - L_2 sign(\sum_{j=1}^{m}(i_j - \hat{i}_j)) \quad (15)$$

where

$$\Delta f_2 = f_2(\theta,i,\omega) - f_2(\hat{\theta},\hat{i},\hat{\omega})$$

By appropriately choosing the two gains L_1 and L_2, we can make $e_\theta \dot{e}_\theta < 0$ and $e_\omega \dot{e}_\omega < 0$. This means $e_\theta \Rightarrow 0$ (or $\hat{\theta} \Rightarrow \theta$) and $e_\omega \Rightarrow 0$ (or $\hat{\omega} \Rightarrow \omega$).

A block diagram of sensorless control using sliding mode observers is shown in figure 2.

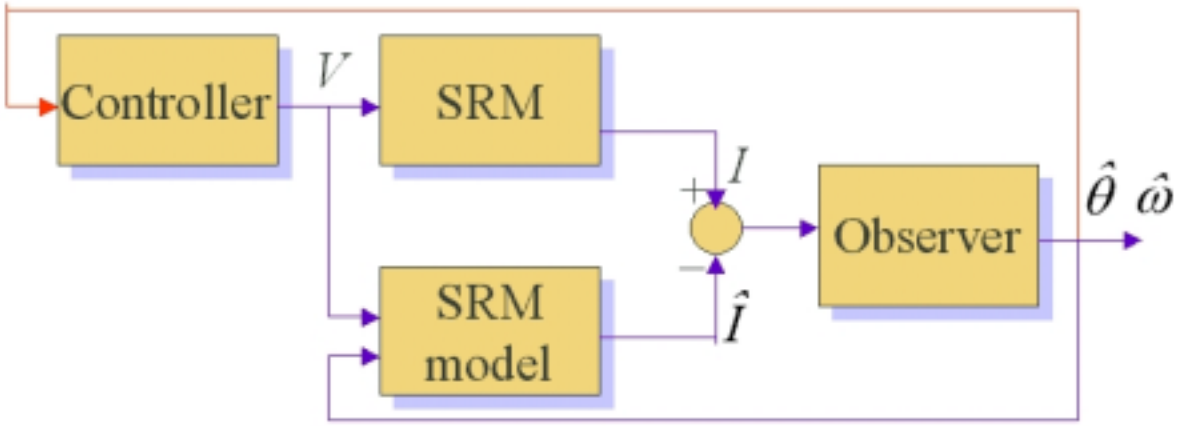


Figure 2. Diagram of sliding mode observer based sensorless controller

A Simulink model of the sensorless control using sliding mode observers is shown in figure 3.

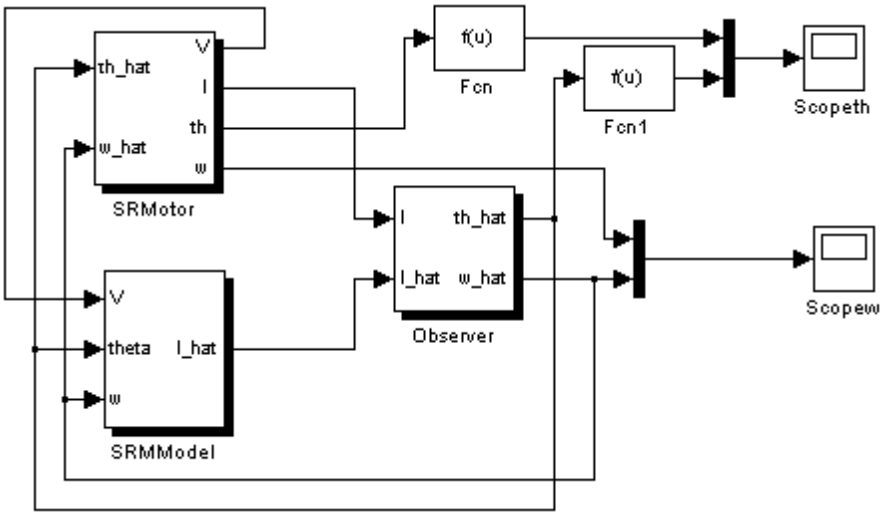


Figure 3. Simulink model of sliding mode observer based sensorless controller

Simulation Results:

The above system is simulated with a time step of T=20μs. The results are shown in figure 4 (1000 RPM) and figure 5 (20 RPM). In each figure, the top two graphs show the actual rotor position (red curve), estimated rotor position (blue curve in above graph), and position estimation error (blue curve in below graph); the bottom two graphs show the actual rotor speed (red curve), estimated rotor speed (blue curve in above graph), and speed estimation error (blue curve in below graph). It is clear that at high speeds, the estimation errors are negligible. But when the speed is below 20 RPM, the error becomes significant, other sensorless control methods must be developed for near zero speeds. This is detailed in next section.

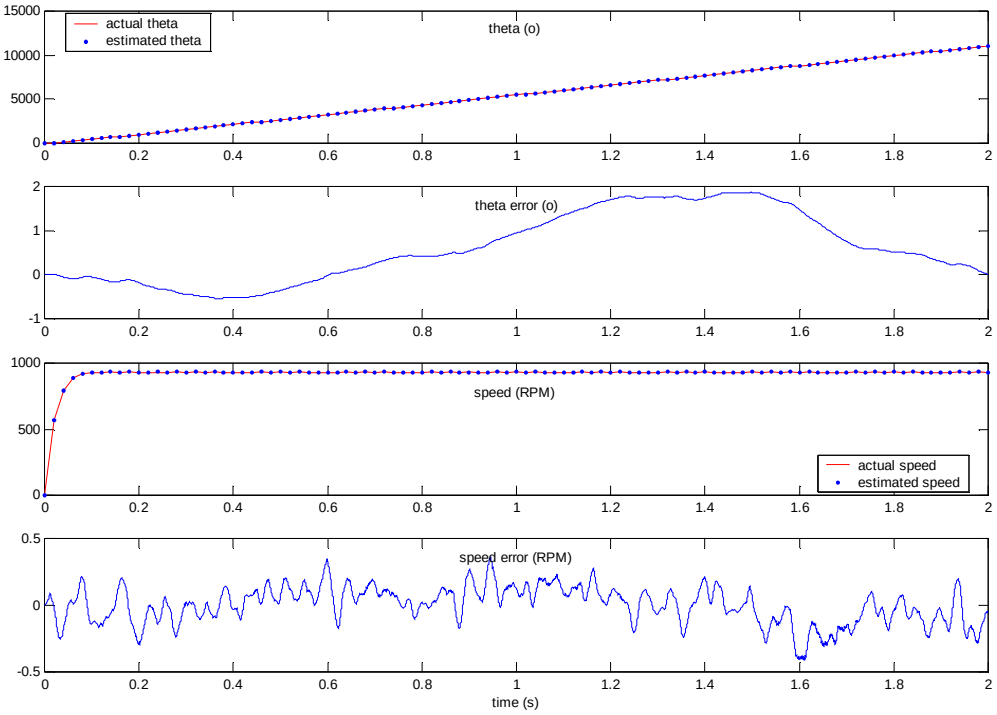


Figure 4. Simulation results at 1000 RPM

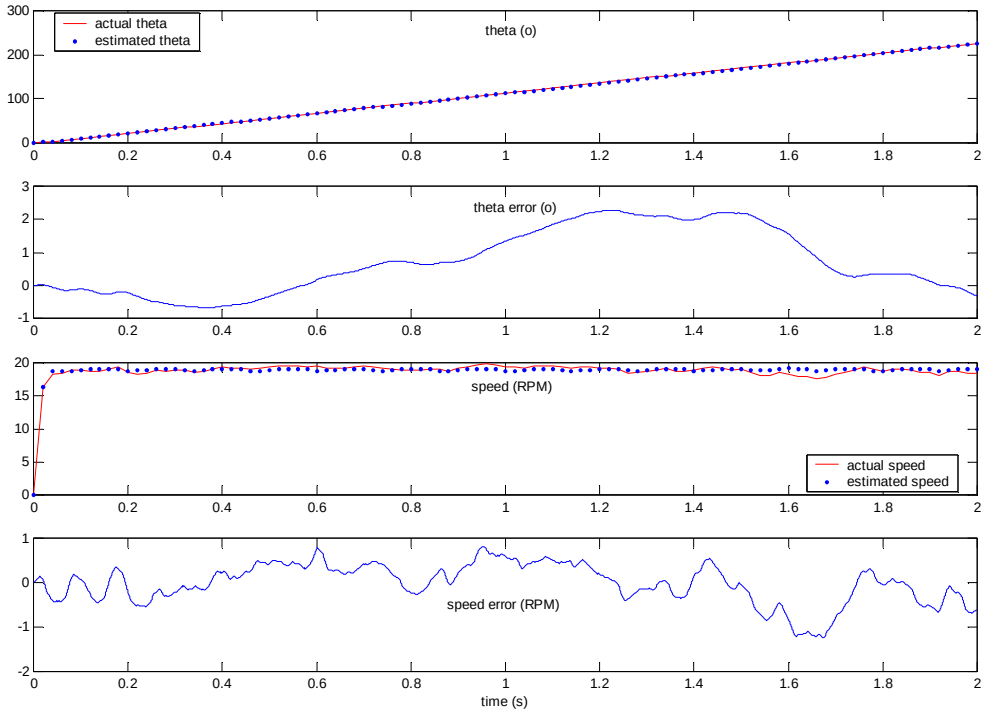


Figure 5. Simulation results at 20 RPM

5. Sensorless Control at Near Zero Speeds

When the rotor speed is low (near zero speeds), the sliding mode observer based controller cannot yield satisfactory results. Other methods must be used for this speed region.

By expanding phase voltage equation (5), we have

$$V_j = R \cdot i_j + (L_j + i_j \frac{\partial L_j}{\partial i_j}) \frac{di_j}{dt} + i_j \frac{\partial L_j}{\partial \theta} \omega . \quad (16)$$

The last item in equation (16) represents the motional back-EMF, which can be ignored at near zero speeds. So we have

$$L_j + i_j \frac{\partial L_j}{\partial i_j} \approx (V_j - R \cdot i_j) / \frac{di_j}{dt} . \quad (17)$$

Let's observe eq. (17). The right side of eq. (17) can be computed directly from terminal measurements V_j and i_j; while the left side contains rotor position information. At different rotor position θ, the left side of eq. (17) will have a different value (with i_j known). So if the value of the right side matches the value of the left side at a given θ, we can know that the rotor reaches the specified position. Based on this observation, we can easily find a way to determine the turn-on/turn-off moment of the SRM.

Let's define

$$F(\theta, i_j) = L_j + i_j \frac{\partial L_j}{\partial i_j} , \text{ and}$$

$$G(V_j, i_j) = (V_j - R \cdot i_j) \frac{\Delta t}{\Delta i_j} .$$

Then when phase *j* is conducting and if

$$F(\theta_{off}, i_j) = G(V_j, i_j) ,$$

that means the turn-off position of phase *j* is arrived and phase *j* should be turned off. Similarly, when phase *j* is conducting and if

$$F(\theta_{on} + \frac{\pi}{2N_r}, i_j) = G(V_j, i_j) ,$$

that means the turn-on position of phase *j+1* is arrived and phase *j+1* should be turned on.

By the way, $F(\theta,i_j)$ can be computed from the inductance model as

$$F(\theta,i)=\frac{1}{2}\left[\frac{1}{2}\left(L_a^*+L_u\right)+L_m^*\right]+\frac{1}{2}\left(L_a^*-L_u\right)\cos N_r\theta+\frac{1}{2}\left[\frac{1}{2}\left(L_a^*+L_u\right)+L_m^*\right]\cos 2N_r\theta$$

where

$$L_a^*(i)=\sum_{n=0}^{k}(n+1)a_n i^n \text{ , and}$$

$$L_m^*(i)=\sum_{n=0}^{k}(n+1)b_n i^n \text{ ,}$$

A Simulink model of the sensorless control at near zero speeds is shown in figure 6.

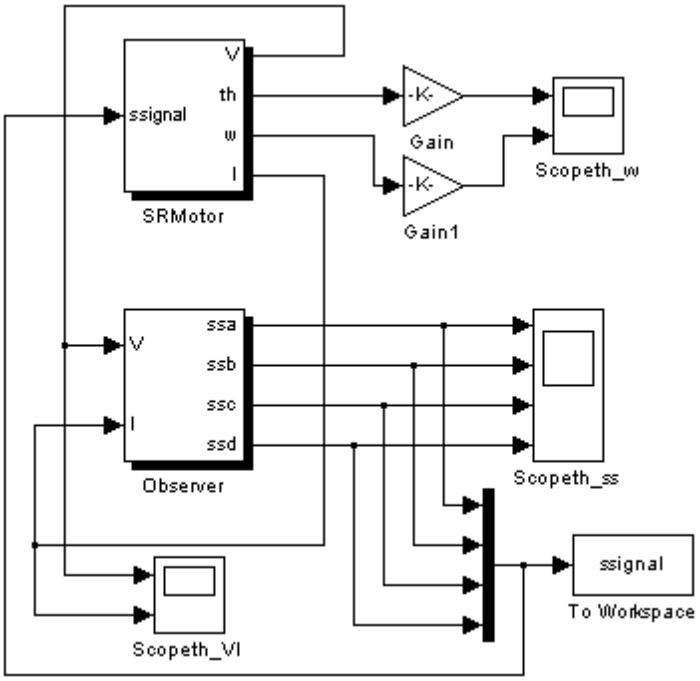


Figure 6. Simulink model of sensorless control at near zero speeds

Simulation Results:

The above system is simulated with a time step of T=20μs. The results are shown in figure 7 (5 RPM). The desired turn-on/turn-off angles and the actual turn-on/turn-off angles (generated by the controller) are listed in table 1. The

angle estimation errors are also listed in the table. It is clear that the error is negligible. Satisfactory results at near zero speeds (less than 20 RPM in our case) can be obtained.

Table 1. Simulation results at near zero speeds

Angle	Desired (o)	Actual (o)	Error (o)
θ_{onA}	-30	-29.49	0.51
θ_{offA}	-7	-6.14	0.86
θ_{onB}	-15	-14.49	0.51
θ_{offB}	8	8.86	0.86
θ_{onC}	0	0.51	0.51
θ_{offC}	23	23.86	0.86
θ_{onD}	15	15.51	0.51
θ_{offD}	38	38.86	0.86

By combining the above two algorithms, a sensorless scheme that yields satisfactory results over full speed range can be obtained.

6. Future Work

Sliding mode observer based sensorless control scheme uses only phase voltages and currents that can be easily measured from motor terminals to estimate rotor speed and position. All functions can be realized by a powerful microprocessors (DSP) and no extra hardware is necessary, which makes it cost-effective. It works very well when speed is above 20 RPM. At near zero speeds, another sensorless algorithm can be used to determine the turn-on/turn-off moment. Simulink simulation shows that satisfactory results can be obtained.

The next step of the research should be field experiments. This includes:

- Setup an experimental testbed with DS1103, SRM, power converters, and accurate position encoder.
- Convert Simulink models into C or assembly code that can be executed on DS1103.
- Test the accuracy of the speed and rotor position observer at full speed range.
- Suggest appropriate filtering algorithm to reduce the effect of noise.
- Test the robustness of the sliding mode observer to the change of motor parameters.

7. References

[1] Khwaja M. Rahman, Babak Fahimi, G. Suresh, Anandan Velayutham Rajarathnam, and M. Ehsani, “Advantages of Switched Reluctance Motor Applications to EV and HEV: Design and Control Issues”, IEEE Transactions

on Industry Applications, Vol. 36, No. 1, January/February, 2000. pp. 111-121

[2] B. Fahimi, G. Suresh, J. Mahdavi, and M. Ehsani, “A New Approach to Model Switched Reluctance Motor Drive Application to Dynamic Performance Prediction, Control and Design”, *Power Electronics Specialists Conference*, 1998. Vol. 2, pp. 2097-2102

[3] Vadim Utkin, Jurgen Guldner, and Jingxin Shi, “Sliding Model Control in Electromechanical Systems”, *Taylor & Francis Inc*., 1999.

[4] Y. J. Zhan, C. C. Chan, and K. T. Chau, “A novel sliding-mode observer for indirect position sensing of switched reluctance motor drives”, *IEEE Transactions on Industrial Electronics*, Vol. 46, Issue 2, April 1999, pp. 390-397

[5] I. Husain, S. Sodhi, and M. Ehsani, “S sliding mode observer based controller for switched reluctance motor drives”, *Conference Recoed of the 1994 IEEE Inductry Application Society Annual Meeting*, Vol. 1, 1994, pp. 635-643

Appendix

- **C Program for Data Acquisition and Over Current Protection**
- **Matlab Program for Alignment of SRM**
- **Matlab Program for Standstill Test**
- **Matlab Programs for Maximum Likelihood Estimation**
- **Matlab Programs for Model Validation**

C Program for Data Acquisition and Over Current Protection – sstest.c

```
/************************************************************
*
* FILE:
*   sstest.c
*
* RELATED FILES:
*   Brtenv.h - Basic Real Time Environment
*
* DESCRIPTION:
*  Standstill test program for SRM using DS1103 controller board.
*
* AUTHOR:
* Wenzhe Lu
* Mechatronics Laboratory, The Ohio State University
* May 12, 2000
*
************************************************************/

#include <Brtenv.h>

/*----------------------------------------------------------*/

#define DT 2.5e-5             /* 25 us simulation step size */
#define NINPUTS 4                     /* number of INPUTS */
#define I_MAX_SET 5                /* peak current rating */
#define I_RMS_SET 3.15       /*square of rms current rating */
#define SCALECURRENT 24.6
#define SCALEVOLTAGE 40

/* variables for TRACE */
Float64 u[NINPUTS] = {0.0, 0.0, 0.0, 0.0};
Float64 i[NINPUTS] = {0.0, 0.0, 0.0, 0.0};
Float64 i_rms[NINPUTS] = {0.0, 0.0, 0.0, 0.0};
Float64 i_oc_rms = 0.0, i_oc_max = 0.0;
UInt32  phase_oc = 0;
UInt32  count[NINPUTS] = {0, 0, 0, 0};
UInt32  bHallSensor, bPrevHallSensor = 0;
UInt32  bRotorPosition;
UInt32  iSample = 0;
Int16   task_id = 0;               /* communication channel */
Int16   index = -1;                  /* command table index */
UInt32  bOverCurrent = 0;              /* over current flag */
UInt32  bPhaseSelect = 1;
Float64 fPulseWidth = 1.0e-4;
UInt32  bPulseOn = 0;
UInt32  bAlign = 0;
UInt32  bRotate = 0;
Float64 exec_time;                       /* execution time */

UInt8  k = 0, m = 1;

/* functions defined in this program */
void isr_timerA(void);
void rms_current_calculation(void);
```

```
void over_current_protection(void);
void rotor_position_detection(void);

/*-----------------------------------------------------------*/

void main(void)
{
 /* init ds1103 */
 ds1103_init();

 /* init communication with slave_dsp */
 ds1103_slave_dsp_communication_init();

/* Initialize Bit 4,6 of group 3 for Bit In: SCAP1 and SCAP3 */
 ds1103_slave_dsp_bit_io_init(task_id, 3,
    SLVDSP1103_BIT_IO_BIT4_MSK | SLVDSP1103_BIT_IO_BIT6_MSK,
    SLVDSP1103_BIT_IO_BIT4_IN | SLVDSP1103_BIT_IO_BIT6_IN );
 /* register read function in the command table for group 3 */
 ds1103_slave_dsp_bit_io_read_register(task_id,
      &index, 3);

 /* init digital I/O ports: set I/O port1 to output */
 ds1103_bit_io_init(DS1103_DIO1_OUT);

 /* turn off all switches */
 ds1103_bit_io_write(0x00000000);

 /* adjust mux 1 (converter 1) to channel 2,
           mux 2 (converter 2) to channel 6,
           mux 3 (converter 3) to channel 10,
           mux 4 (converter 4) to channel 14, */
 ds1103_adc_mux_all(2, 6, 10, 14);

 /* start AD convertion */
 ds1103_adc_delayed_start(DS1103_ADC1 | DS1103_ADC2 |
                          DS1103_ADC3 | DS1103_ADC4);

 /* wait 5 us for AD convertion*/
 ds1103_tic_delay(5.0e-6);

 /* periodic event in ISR */
 ds1103_start_isr_timerA(DT, isr_timerA);

 /* Background task */
 while(1)
 {
    master_cmd_server();
    host_service(0, 0);                  /* COCKPIT service */
    if ((bPulseOn == 1) & (bOverCurrent == 0))
    {
        switch (bPhaseSelect)
        {
         case 1:
            ds1103_bit_io_write(0x0000000a);
            ds1103_tic_delay(fPulseWidth);
            ds1103_bit_io_write(0x00000000);
            break;
```

```
     case 2:
        ds1103_bit_io_write(0x000000a0);
        ds1103_tic_delay(fPulseWidth);
        ds1103_bit_io_write(0x00000000);
        break;
     case 3:
        ds1103_bit_io_write(0x00000005);
        ds1103_tic_delay(fPulseWidth);
        ds1103_bit_io_write(0x00000000);
        break;
     case 4:
        ds1103_bit_io_write(0x00000050);
        ds1103_tic_delay(fPulseWidth);
        ds1103_bit_io_write(0x00000000);
        break;
     default:
        ds1103_bit_io_write(0x00000000);
    }
    bPulseOn = 0;
}
if ((bAlign == 1) || (bRotate == 1))
{
    if (((k < 1000) || (bRotate == 1)) && (bOverCurrent == 0))
    {
        if ((i[bPhaseSelect-1] < 2.5) & (m == 1))
        {
            switch (bPhaseSelect)
            {
            case 1:
                ds1103_bit_io_write(0x0000000a);
                break;
             case 2:
                ds1103_bit_io_write(0x000000a0);
                break;
             case 3:
                ds1103_bit_io_write(0x00000005);
                break;
             case 4:
                ds1103_bit_io_write(0x00000050);
                break;
            }
            m = 0;
            k++;
        }
        else if ((i[bPhaseSelect-1] > 2.9) & (m == 0))
        {
            ds1103_bit_io_write(0x00000000);
            m = 1;
            k++;
        }
    }
    else
    {
        bAlign = 0;
        bRotate = 0;
        ds1103_bit_io_write(0x00000000);
        k = 0;
```

```
            m = 1;
        }
    }
    else
    {
        k = 0;
        m = 1;
    }
 }
}

/*-------------------------------------------------------------*/

void isr_timerA(void)
{
 ds1103_begin_isr_timerA();               /* overload check */
 host_service(1, 0);                       /* TRACE service */

 ds1103_tic_start();              /* start time measurement */

 /* samples current when iSample = 0; else samples voltage  */
 if (iSample == 0)
 {
    iSample = 1;

    /* read results of last convertion */
    ds1103_adc_read2(1, &i[0], &i[1]);
    ds1103_adc_read2(3, &i[2], &i[3]);

    i[0] = i[0] * SCALECURRENT;
    i[1] = i[1] * SCALECURRENT;
    i[2] = i[2] * SCALECURRENT;
    i[3] = i[3] * SCALECURRENT;

    /* adjust mux 1 (converter 1) to channel 4,
              mux 2 (converter 2) to channel 8,
              mux 3 (converter 3) to channel 12,
              mux 4 (converter 4) to channel 16, */
    ds1103_adc_mux_all(4, 8, 12, 16);
 }
 else
 {
    iSample = 0;

    /* read results of last convertion */
    ds1103_adc_read2(1, &u[0], &u[1]);
    ds1103_adc_read2(3, &u[2], &u[3]);

    u[0] = u[0] * SCALEVOLTAGE;
    u[1] = u[1] * SCALEVOLTAGE;
    u[2] = u[2] * SCALEVOLTAGE;
    u[3] = u[3] * SCALEVOLTAGE;

    /* adjust mux 1 (converter 1) to channel 2,
              mux 2 (converter 2) to channel 6,
              mux 3 (converter 3) to channel 10,
              mux 4 (converter 4) to channel 14, */
```

```
    ds1103_adc_mux_all(2, 6, 10, 14);
 }

 /* start AD convertion */
 ds1103_adc_delayed_start(DS1103_ADC1 | DS1103_ADC2 |
                          DS1103_ADC3 | DS1103_ADC4);

 if (iSample == 1)
    rms_current_calculation();
 else
 {
    over_current_protection();
    rotor_position_detection();
 }

 exec_time = ds1103_tic_read();

 ds1103_end_isr_timerA();
}

/*-------------------------------------------------------------*/
void rms_current_calculation(void)
{
   UInt8    j;
   for (j=0; j<NINPUTS; j++)
   {
      i_rms[j] = sqrt((i_rms[j] * i_rms[j] * (Float64)count[j] + i[j] *
i[j]) / (Float64)(count[j] + 1));
      count[j]++;
   }
}

/*-------------------------------------------------------------*/
void over_current_protection(void)
{
   phase_oc = 0;
   if ((i[0] > I_MAX_SET) | (i_rms[0] > I_RMS_SET))
        phase_oc = 1;
   else if ((i[1] > I_MAX_SET) | (i_rms[1] > I_RMS_SET))
        phase_oc = 2;
   else if ((i[2] > I_MAX_SET) | (i_rms[2] > I_RMS_SET))
        phase_oc = 3;
   else if ((i[3] > I_MAX_SET) | (i_rms[3] > I_RMS_SET))
        phase_oc = 4;

   if (phase_oc != 0)
   {
      /* turn off all switches */
      ds1103_bit_io_write(0x00000000);

      /* record fault current */
      i_oc_rms = i_rms[phase_oc-1];
      i_oc_max = i[phase_oc-1];

      /* set over current flag */
      bOverCurrent = 1;
      bPulseOn = 0;
```

```
      bAlign = 0;
   }
}

/*-------------------------------------------------------------*/
void rotor_position_detection(void)
{
   UInt8 temp;
   Int16 slave_err_read;
   UInt8 H1, H2, pH1;
   UInt8 j;

   /* request a read from digital I/O port */
   ds1103_slave_dsp_bit_io_read_request(task_id, index);

   /* read bitmap from the specified I/O group */
   do
   {
      slave_err_read = ds1103_slave_dsp_bit_io_read(task_id,
         index, &temp );
   }
   while(slave_err_read == SLVDSP1103_NO_DATA);

   bHallSensor = temp & 0x50;   /* 01010000 */

   /* determine rotor position if the Hall sensor gives new signal */
   if (bPrevHallSensor != bHallSensor)
   {
      H1 = ((bHallSensor & 0x10) == 0x10);
      H2 = ((bHallSensor & 0x40) == 0x40);
      pH1 = ((bPrevHallSensor & 0x10) == 0x10);
      if (pH1 == H2)
      {
         bRotorPosition =
H1*H2+H1*(H2==0)*2+(H1==0)*(H2==0)*3+(H1==0)*H2*4;
      }
      else
      {
         bRotorPosition =
H1*(H2==0)+(H1==0)*(H2==0)*2+(H1==0)*H2*3+H1*H2*4;
      }

      if (bRotate == 1)
      {
         ds1103_bit_io_write(0x00000000);
         bPhaseSelect = bRotorPosition;
         k = 0;
         m = 1;
      }
      bPrevHallSensor = bHallSensor;
      for (j=0; j<NINPUTS; j++)
      {
         count[j] = 0;
      }
   }
}
```

Matlab Program for Alignment of SRM – align.m

```
function align(phase)
% Syntax:    Standstill Test of SRM Parameters;
%
% Purpose:   Estimate SRM parameters through stnadstill test.
%
% This file needs the real-time processor application 'sstest.ppc'
running.
% To change simulation parameters while the continuous data acquisition
% is running open ControlDesk and load the experiment file
'sstest.cdx'.

% Copyright (c) 2000 by OSU


% select DS1103 board for use with MLIB
mlib('SelectBoard','DS1103');

% check if the application sstest.ppc is running
ApplName = lower([pwd,'\sstest.ppc']);
if mlib('IsApplRunning'),
   ApplInfo = mlib('GetApplInfo');
   if strcmp(ApplName,lower(ApplInfo.name)) ~= 1
      err_msg = sprintf('*** This MLIB demo file needs the real-time
processor application\n*** ''%s'' running!',...
                        ApplName);
      error(err_msg);
   end;
else
   err_msg = sprintf('*** This MLIB demo file needs the real-time
processor application\n*** ''%s'' running!',...
                        ApplName);
   error(err_msg);
end;

% specify the variables used by MLIB and get they descriptors
variables = {'Signals/i[0]'; 'Signals/u[0]'; 'Signals/i[1]';
'Signals/u[1]';
             'Signals/i[2]'; 'Signals/u[2]'; 'Signals/i[3]';
'Signals/u[3]'};
parameter1 = {'Control Variables/bAlign'};
parameter2 = {'Control Variables/fPulseWidth'};
parameter3 = {'Control Variables/bPhaseSelect'};
parameter4 = {'Control Variables/bRotorPosition'};
parameter5 = {'Control Variables/bPulseOn'};
parameter6 = {'Control Variables/bOverCurrent'};
parameter7 = {'Signals/i_oc_rms'};
parameter8 = {'Signals/i_oc_max'};
parameter9 = {'Signals/phase_oc'};

[Ia, Va, Ib, Vb, Ic, Vc, Id, Vd] = mlib('GetTrcVar', variables);
bAlign = mlib('GetTrcVar', parameter1);
fPulseWidth = mlib('GetTrcVar', parameter2);
bPhaseSelect = mlib('GetTrcVar', parameter3);
Phase = mlib('GetTrcVar', parameter4);
```

```
bPulseOn = mlib('GetTrcVar', parameter5);
bOverCurrent = mlib('GetTrcVar', parameter6);
i_oc_rms = mlib('GetTrcVar', parameter7);
i_oc_max = mlib('GetTrcVar', parameter8);
phase_oc = mlib('GetTrcVar', parameter9);

pw = 2e-1;

% set the option of the data acquisition performed by service number 1
(default)
mlib('Set','StepSize',2.5e-5,...             % continuos mode can be
actived only with disabled triggering
   'TraceVars',[Ia; Va; Ib; Vb; Ic; Vc; Id; Vd],...
   'Start',0,...
   'Stop',pw*3,...
   'Downsampling',2);

clf;
t = [0:5e-5:pw*3+5e-5];
mlib('Write', bOverCurrent, 'Data', 0);
mlib('Write', i_oc_rms, 'Data', 0.0);
mlib('Write', i_oc_max, 'Data', 0.0);
mlib('Write', bPhaseSelect, 'Data', phase);
mlib('StartCapture');     % start the continuous data acqusition on the
DS1103, default service number 1
mlib('Write', bAlign, 'Data', 1);

while mlib('CaptureState') ~= 0
end

out_data = mlib('FetchData');
Ia = out_data(2*phase-1,:);
Va = out_data(2*phase,:);

mlib('StopCapture');     % stop the continuous data acquisition on the
DS1103 board
mlib('Write', bAlign, 'Data', 0);
%mlib('Write', bPulseOn, 'Data', 1);

bOC = mlib('Read', bOverCurrent);
if (bOC == 1)
{

fprintf('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n');
   fprintf('Phase %d over curent!\n', mlib('Read', phase_oc)+1);
   fprintf('RMS current = %f, peak current = %f\n', mlib('Read',
i_oc_rms), mlib('Read', i_oc_max));

fprintf('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n');
};
else
{
   fprintf('=======================================================\n');
   fprintf('Phase %d is in its ALIGNED position now!\n', mlib('Read',
Phase));
   fprintf('\n');
   fprintf('Use sstest to perform standstill test.\n');
```

```
   fprintf('=======================================================\n');
};
end

subplot(2,1,1);
plot(t, out_data(2*phase-1,:), 'b');
ylabel('I');
grid on
subplot(2,1,2);
plot(t, out_data(2*phase,:), 'b');
ylabel('V');
grid on
```

Matlab Program for Standstill Test – sst.m

```
function sst(phase)
% Syntax:    Standstill Test of SRM Parameters;
%
% Purpose:   Estimate SRM parameters through stnadstill test.
%
% This file needs the real-time processor application 'sstest.ppc'
running.
% To change simulation parameters while the continuous data acquisition
% is running open ControlDesk and load the experiment file
'sstest.cdx'.

% Copyright (c) 2000 by OSU


% select DS1103 board for use with MLIB
mlib('SelectBoard','DS1103');

% check if the application sstest.ppc is running
ApplName = lower([pwd,'\sstest.ppc']);
if mlib('IsApplRunning'),
   ApplInfo = mlib('GetApplInfo');
   if strcmp(ApplName,lower(ApplInfo.name)) ~= 1
      err_msg = sprintf('*** This MLIB demo file needs the real-time
processor application\n*** ''%s'' running!',...
                        ApplName);
      error(err_msg);
   end;
else
   err_msg = sprintf('*** This MLIB demo file needs the real-time
processor application\n*** ''%s'' running!',...
                        ApplName);
   error(err_msg);
end;

% specify the variables used by MLIB and get they descriptors
variables = {'Signals/i[0]'; 'Signals/u[0]'; 'Signals/i[1]';
'Signals/u[1]';
             'Signals/i[2]'; 'Signals/u[2]'; 'Signals/i[3]';
'Signals/u[3]'};
parameter1 = {'Control Variables/bPulseOn'};
parameter2 = {'Control Variables/fPulseWidth'};
parameter3 = {'Control Variables/bPhaseSelect'};
parameter4 = {'Control Variables/bRotorPosition'};
parameter5 = {'Control Variables/bOverCurrent'};
parameter6 = {'Signals/i_oc_rms'};
parameter7 = {'Signals/i_oc_max'};
parameter8 = {'Signals/phase_oc'};

[Ia, Va, Ib, Vb, Ic, Vc, Id, Vd] = mlib('GetTrcVar', variables);
bPulseOn = mlib('GetTrcVar', parameter1);
fPulseWidth = mlib('GetTrcVar', parameter2);
bPhaseSelect = mlib('GetTrcVar', parameter3);
Phase = mlib('GetTrcVar', parameter4);
bOverCurrent = mlib('GetTrcVar', parameter5);
```

```
i_oc_rms = mlib('GetTrcVar', parameter6);
i_oc_max = mlib('GetTrcVar', parameter7);
phase_oc = mlib('GetTrcVar', parameter8);

pw = 150e-4;

% set the option of the data acquisition performed by service number 1
(default)
mlib('Set',...              % continuos mode can be actived only with
disabled triggering
   'TraceVars',[Ia; Va; Ib; Vb; Ic; Vc; Id; Vd],...
   'NumSamples',fix(2*pw/5e-5)+1,...
   'Downsampling',2);

clf;
t = [0:5e-5:2*pw];
mlib('Write', bOverCurrent, 'Data', 0);
mlib('Write', i_oc_rms, 'Data', 0.0);
mlib('Write', i_oc_max, 'Data', 0.0);
mlib('Write', fPulseWidth, 'Data', pw);
%phase = 1;
mlib('Write', bPhaseSelect, 'Data', phase);
mlib('StartCapture');     % start the continuous data acqusition on the
DS1103, default service number 1
mlib('Write', bPulseOn, 'Data', 1);

while mlib('CaptureState') ~= 0
end

out_data = mlib('FetchData');
I = out_data(2*phase-1,:);
V = out_data(2*phase,:);

mlib('StopCapture');     % stop the continuous data acquisition on the
DS1103 board
mlib('Write', bPulseOn, 'Data', 0);

save VI.mat V I

h = 1;
t = length(I);
while (h < length(I))
   if abs(I(h)) > 0.15
      break;
   else h = h+1;
   end
end
while (t > 1)
   if abs(I(t)) > 0.2
      break;
   else t = t-1;
   end
end
V = V(h:t);
I = I(h:t);

subplot(2,1,1);
```

```
plot(I);
axis([0 length(I) -1 6]);
ylabel('I');
grid on
subplot(2,1,2);
plot(V);
axis([0 length(V) -25 25]);
ylabel('V');
grid on

Z = [I' V'];

R = 0.9;
L = 1e-3;

th_orig = [R L];

options(14) = 2000;

[th, options] = leastsq('mtfun', th_orig, options, [], Z);

R = th(1)
L = th(2)
```

Matlab Programs for Maximum Likelihood Estimation – mle.m

```
s = length(I);

h = 1;
while (h < s)
   if abs(I(h)) > 0.15
      break;
   else h = h+1;
   end
end
t = s;
while (t > 1)
   if abs(I(t)) > 0.2
      break;
   else t = t-1;
   end
end
V = V(h-1:t+2);
I = I(h-1:t+2);


% Get input and measured output
Z = [I' V'];

% Guess initial parameters
R = 0.9;
L = 0.01;

th = [R L];

% set leastsq() options
options(1)  = 1;     % display intermediate result
options(14) = 2000;  % maximum iteration count

% maximum likelihood estimation: Kalman filtering + OE
% estimate parameters using output error estimation - leastsq()
% Kalman filter is used inside errfun()
[th, options] = leastsq('errfun', th, options, [], Z);

R = th(1)-7.275284;
L = th(2);

% display results
fprintf('\n=================================================\n');
fprintf(' Estimated parameters: \n');
fprintf('   R = %9.6f ohm\n', R);
fprintf('   L = %9.6f H\n', L);
fprintf('=================================================\n');

validate;
```

Matlab Programs for Maximum Likelihood Estimation – errfun.m

```
function err = errfun(th, Z)
% This function is called by leastsq() function.
% It is used to compute the estimation err e=y-yhat in each iteration.

% covariance of process noise and measurement noise
Q = 1e-10;
R = 1e-2;

% Input and output
I = Z(:,1);
V = Z(:,2);

% Parameters
RR  = th(1);
LL  = th(2);

% Form the state space matrix from parameters
Ac = -inv(LL) * RR;
Bc = inv(LL)';

% Convert to discrete-time model
T = 5e-5;
[Ad, Bd] = c2d(Ac, Bc, T);
Cd = 1;

% Design Kalman filter
sys = ss(Ad, [Bd 1], Cd, 0, -1);
[kalmf,L,P,M] = kalman(sys, Q, R);
kalmf = kalmf(1,:);

% Estimate output through Kalman filter using input and measured output
ZZ = [V I];
yhat = lsim(kalmf, ZZ);

% compute estimation error: e = y - yhat
err = I - yhat;
```

Matlab Programs for Model Validation – validate.m

```
x0 = [0.0];

y = I;

RR  = th(1);
LL  = th(2);

Ac = -RR/LL;
Bc = 1/LL;

T = 5e-5;

[Ad, Bd] = c2d(Ac, Bc, T);

U = V;
xhat = ltitr(Ad, Bd, U');
yhat = xhat;

cove = cov(y-yhat')
stde = std(y-yhat')

t = 5e-5 * (1:length(U));
subplot(2,1,1)
plot(t, yhat', 'cx', t, y, 'r-')
title('I (A)');
legend('actual', 'estimated');
axis([0.0 5e-5*length(U) 0 3]);
grid
subplot(2,1,2)
plot(t, y-yhat', 'r')
title('error (A)');
xlabel('time (s)');
axis([0 5e-5*length(U) -1 1]);
grid
```