AN2516
Application note

Luenberger state observer
Rotor position estimation simulink and software library

Introduction

This application note describes a software library for estimating the rotor position of a 3 phase permanent magnet synchronous motor (PMSM) using a Luenberger state observer. It is also shown how to use a luenberger state observer in a flux oriented control (FOC) scheme to implement a sensorless vector control strategy.

Contents

List of tables

List of figures

1 Introduction

This document describes a software library for estimating the rotor position of a 3 phase permanent magnet synchronous motor (PMSM) using a Luenberger state observer. The luenberger state observer used in a Flux oriented control strategy allows to implement sensorless vector control strategy.

The library consists of:

- Simulink library;
- Software library.

The simulink library consists of a set of functions for implementing in Matlab-Simulink environment the Luenberger state observer to estimate the back emf from which is possible to calculate the motor rotor position. These can be used as base blocks to conceive and to test new electric motor sensorless controls and to produce automatic generated code in ANSI C, downloadable on microcontroller.

The software library is a set of routines for the rotor position estimation implemented for ST10 microcontrollers obtained from the code generated in automatic, starting from simulink library, and then optimized in assembler. The software library is equivalent to the simulink library, from point of view of bit accuracy, same API.

This document, after a brief introduction on the permanent magnet synchronous machine (PMSM), describes the proposed sensorless strategy, the rotor position estimation from back emf and the Luenberger state observer. Then how to use the state observer in the flux oriented Control (FOC) sensorless strategy is shown.

Finally the simulink library and the software library follow.

2 PMSM motors

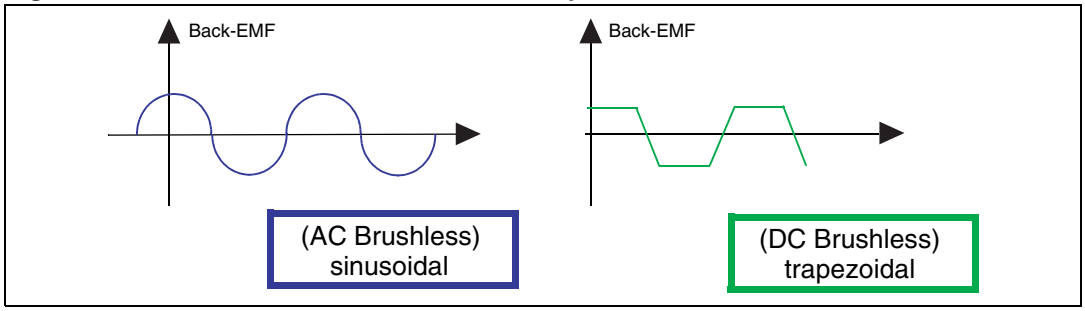
Brushless permanent magnet motors (PMSMs) are electric motors with the same electromagnetic structure of a synchronous machine but without the brushes. They have a wound stator, similar to an induction machine, and a permanent magnet rotor that replaces a rotor fed with dc current, like a synchronous machine. The PMSMs are not self-commuting motors and to produce useful torque, the currents and the voltages applied to stator phases must be controlled as a function of rotor position. Therefore it is generally required to count the rotor position with a sensor, like Hall sensors, encoder or resolver, so that the inverter phases which feed it, acting at any time, are commuted depending on the rotor position.

PMSMs are usually classified as:

- trapezoidal (DC Brushless);
- sinusoidal (AC Brushless).

According to the waveform of the voltage induced by the rotor magnet in the stator phases, the so-called “back-EMF”, which is determined by the winding distribution and the rotor shape.

Figure 1. PMSMs back-EMF waveform shapes



Depending on the motor type (trapezoidal or sinusoidal) different control algorithms are implemented:

- by square wave currents and two phases on operation (DC Brushless technique);
- by sinusoidal wave currents and three phases on operation (AC Brushless technique).

These algorithms affect hardware requirements, sensorless strategy and overall drive cost.

The FOC control method is often applied to PMSM motors with a sinusoidal back-EMF waveform shape to achieve high torque performance and efficiency.

3 State observer sensorless strategy

A state observer based on sensorless control strategy is a good solution for a wide range of fixed speed and low cost applications such as fuel pumps or fans.

In a state observer the complete differential motor model is used to estimate the whole state variable which include both the (unknown) rotor speed and position and the (measurable) motor currents. The observer needs relative accuracy in the modeling of the equation of the unknown variables, the measurements of the motor currents, and the knowledge of the feeding voltages. The instantaneous error between the measured and estimated motor currents are used to adjust the estimation of the unknown variables.

This approach has excellent performance in medium/high speed application while has problems at low or standstill operation when the back emf is low.

Many are the advantages of the sensorless drives:

- lower cost,
- reduced size of the drive machine,
- elimination of the speed sensor cable and increased reliability.

while the existence of shaft-mounted sensors inherently adds some drawbacks to the PM motor drive system, such as:

- An increase in the number of connections between the motor and the control system.
- Interference increases.
- Limitations in accuracy of the sensor because of environmental factors such as temperature, humidity, vibration (decrease the reliability of the system).
- Additional system cost (specially, for small power motors the effect is very high).
- It complicates the design of the motor, especially in PM brushless DC motors, because of the requirement of mounting of the sensor devices inside the motor housing.

4 Rotor position estimation

The classical solutions for rotor position detection rely on its dependency from the back electromotive force (EMF) inducted in the stator windings.

Assuming **isotropic, star connected PMSM motors** having sinusoidal shaped back-EMF waveforms, the detection of the rotor position can be done on basis of the DQ components of the motor back-EMF (e_D, e_Q).

In general the dependency of θ_r from back-EMF can be described by *Equation 1*:

Equation 1

$$e_{DQ} = (k_e \cdot w_r \cdot f_{DQ}(\theta_r))$$

where:

e_{DQ}	back-EMF symmetrical components
k_e	back-EMF constant
w_r	rotor speed
f_{DQ}	DQ shape functions
θ_r	rotor angle

In particular for a PMSM with back emf sinusoidal, there is the following relationship between rotor position and bemf:

Equation 2

$$e_{D1}(\theta_r) = -k_e \cdot w_r \cdot \sin(\theta_r)$$

Equation 3

$$e_{Q1}(\theta_r) = k_e \cdot w_r \cdot \cos(\theta_r)$$

The rotor position is obtained after inverting the trigonometric functions $\sin(\theta_r)$ and $\cos(\theta_r)$.

Equation 4

$$\theta_r = \operatorname{acos}\left(\frac{e_{Q1}}{\sqrt{e_{D1}^2 + e_{Q1}^2}}\right)$$

5 Luenberger observer (LO)

Let's consider the PMSM motor voltage equations in the stator frame:

Equation 5

$$v_D = R_s \cdot i_D + L_s \cdot \frac{di_D}{dt} + e_D$$

Equation 6

$$v_Q = R_s \cdot i_Q + L_s \cdot \frac{di_Q}{dt} + e_Q$$

in order to set up a back emf observer, the induced back emf components, $[e_D\ e_Q]$, can be considered as disturbance with the following associated model:

Equation 7

$$\frac{de_D}{dt} = 0$$

Equation 8

$$\frac{de_Q}{dt} = 0$$

from the eq [1.5] [1.6].[1.7] [1.8] extended PMSM model is obtained:

Equation 9

$$\hat{x}_E = A_E \cdot \hat{x}_E + B_E \cdot u$$

where

Equation 10

$$\hat{x}_E = [i_D, i_Q, e_D, e_Q]^T$$

vector state variables

Equation 11

$$u = [v_D, v_Q]^T$$

input vector

and A_E, B_E, C_E are the matrices of system parameters:

$$A_E = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} \qquad B_E = \begin{bmatrix} B \\ 0 \end{bmatrix} \qquad C_E = \begin{bmatrix} C & 0 \end{bmatrix}$$

$$A = -\frac{R_s}{L} \qquad B = -\frac{1}{L} \qquad C = I$$

while ***I*** and *0* (R^{2x2}) are the "identity" and "zero" sub-matrices respectively.

From the extended model, the Luenberger observer for back emf is obtained as follow:

Equation 12

$$\hat{x}_E = A_E \cdot \hat{x}_E + B_E \cdot u + K \cdot (y - (C_E \cdot \hat{x}_E))$$

where

Equation 13

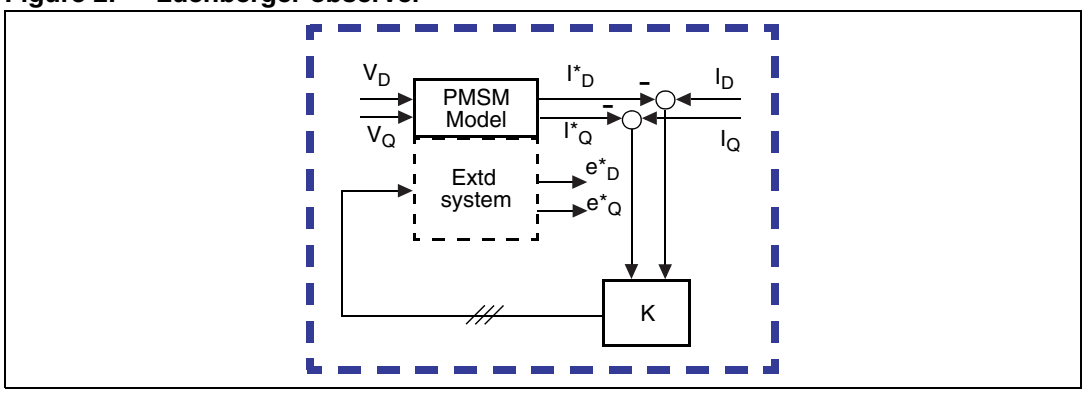
$$y = [i_D, i_Q]^T$$ **output vector**

K the observer gain matrix

It is characterized by an equation containing a term that corrects the current state estimates by an amount proportional to the prediction error: the estimation of the current output minus the actual measurement. This correction ensures stability and convergence of the observer even when the system being observed is unstable.

The Luenberger observer is used to detect the instantaneous value of the two motor back-EMF components, in the stator frame, of a AC brushless motor needed to identify the rotor position.

Figure 2. Luenberger observer



6 Sensorless control scheme

In flux oriented control, motor currents and voltages are manipulated in the d-q reference frame of the rotor. This means that measured motor currents must be mathematically transformed from the three-phase stationary reference frame (a,b,c) of the stator windings to the two axis rotating d-q reference frame, prior to processing, for example by PI controllers (it is possible to use a different controller). Similarly, the voltages to be applied to the motor are mathematically transformed from d-q frame of the rotor to the three phases reference frame of stator before they can be used to produce the voltage control signals for the output inverter that feeds the motor.

These transformations are the core of flux oriented control.

Simplifying the expression of the electrical model of the machine, the projection from the three-phase stationary reference frame of the stator windings to the two axis rotating reference frame can be executed into two subsequent steps:

- (a,b,c) => (D,Q) (the clarke transformation) which outputs a two co-ordinate time variant system;
- (D,Q) => (d,q) (the park transformation) which outputs a two co-ordinate time invariant system;

To carry out the projection from a frame to another it is necessary to know in any time the values of the currents in the stator phases and the rotor position estimated by the Luenberger state observer.

Figure 3 shows the block diagram of the FOC control: the two motor phase currents, (i_{s1}, i_{s2}), are measured with two current sensors (e.g. by phase shunts or current transducers) , and then the 2 currents are projected with clarke transformation (forward clarke) in the stator frame *D,Q* . Outputs of this block are the two current components (i_{sD}, i_{sQ}) in the D,Q stator fixed frame.

i_{sD} and i_{sQ} are used inside the Luenberger observer together with v_D v_Q; the output is cos(theta) sen(theta) used for park transformations. These current components are also used as inputs of the park transformation module, (forward park), that gives as output the current components (i_{sd}, i_{sq}) in the *d,q* rotating reference frame.

i_{sd} and i_{sq} measured current components are compared to the references i_{sdref} (the flux reference) and i_{sqref} (the torque reference) and corrected by mean of two PI controllers.

As in brushless synchronous permanent magnet motor the magnet flux is fixed (depending on magnets), in the PMSM control, i_{sdref} should be set to zero, being the only current component in able to weak the flux, while the torque command i_{sqref} could be the output of the speed regulator, e.g. for a speed-FOC. That forces the current space vector $\bar{i}_s$ to be exclusively in the quadrature direction, respect on the magnet flux vector. Since only i_{sq} produces useful torque, this maximizes the torque efficiency of the system.

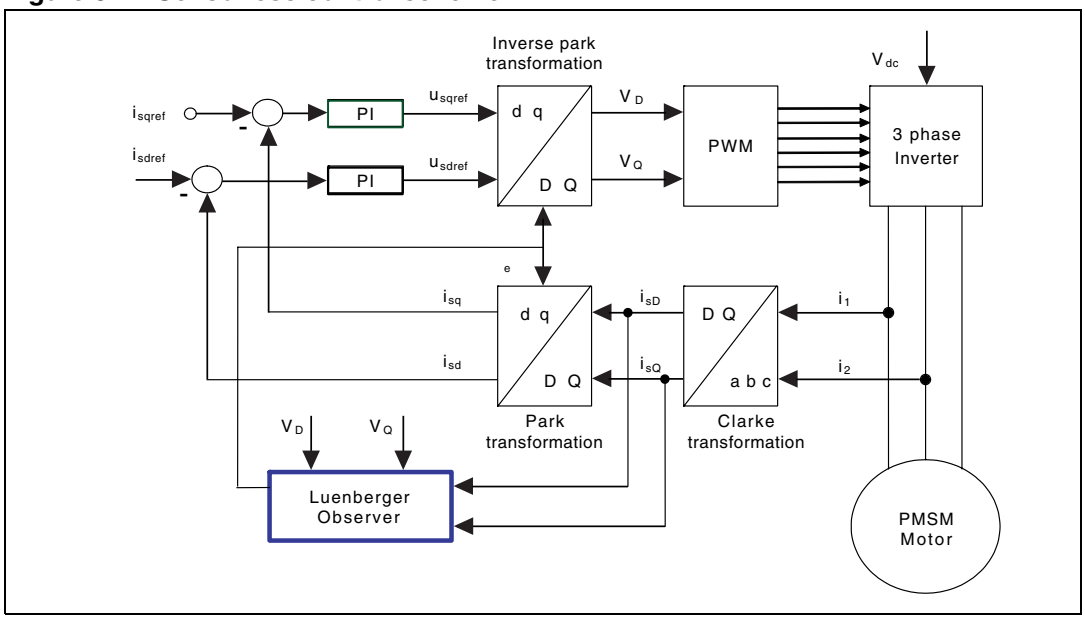
Then the outputs of two PI, u_{sd} and u_{sq}, are sent to the Inverse Park transformation module, (Reverse Park), from which we get the new components of the stator voltage vector in the (u_{sD}, u_{sQ}) non-rotating stator frame.

These signals are then appropriately processed to produce voltage signals for the output bridge.

In our case, it is chosen to use the **space vector modulation (SVM)** technique to impress the new voltage vector to the motor.

The implemented sensorless control scheme is shown in *Figure 3*.

Figure 3. Sensorless control scheme



7 Simulink library

7.1 Description

The Simulink library implements two functions for the sensorless rotor position estimation:

- SineCosine;
- LObserver;

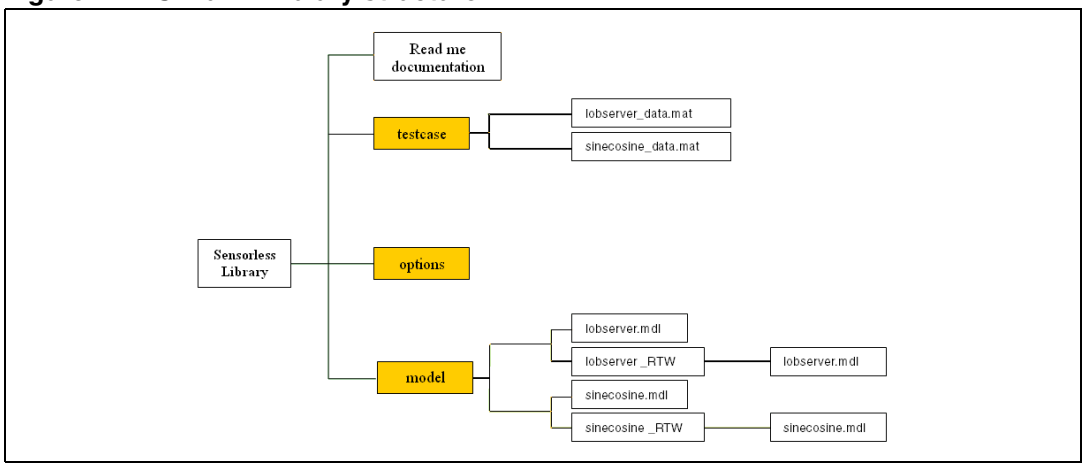
7.2 Using the simulink library

Two are the main directories of the simulink package library:

- 1 directory for all test cases: 1 subdirectory per library function;
- 1 directory for all .mdl files.

The folder structure isshown in *Figure 4*

Figure 4. Simulink library structure



7.2.1 How to install simulink library

The simulink library is delivered as an archive file with .zip extension. To install it, it is necessary to unzip the file in the (C:) directory for a correct use.

Note: You must have a 7.0.0 Matlab version or upward installed on your system to use this library, plus a licence for fixed-point-precision toolbox to use the "convert block"in each scheme block and a licence of RTW embedded coder toolbox.

Please, read the README.txt file in the archive file for using the library.

7.2.2 Test environment

.mat: the inputs and outputs data of the functions obtained by Simulink in the double format are stored. When the mdl file is opened, data is loaded into the workspace of Matlab.

The name of each test-file begins with the (yyy) function name to which it refers, followed by an underscore and the suffix "data".

7.3 Parameters format

The Luenberger state observer in Simulink has the same behavior of that implemented on micro where it is necessary to use a different fixed point precision number representation in every block. In the *Table 1* the variables and their representations are listed:

Table 1. Data representation

Variable	Representation	Description
i_{sD}	sfix(16,8)	direct-axis current component in stator fixed frame
i_{sQ}	sfix(16,8)	quadrature-axis current component in stator fixed frame
cos_t	sfix(16,14)	$\cos(\theta_e)$
sin_t	sfix(16,14)	$\sin(\theta_e)$
usD	sfix(16,6)	direct-axis voltage component in stator frame
usQ	sfix(16,6)	quadrature-axis voltage component in stator frame
usD_old	sfix(16,6)	direct-axis voltage component in stator frame
usQ_old	sfix(16,6)	quadrature-axis voltage component in stator frame
usD_e_next	sfix(16,6)	direct-axis voltage component in stator frame
usQ_e_next	sfix(16,6)	quadrature-axis voltage component in stator frame
k_1	sfix(16,13)	LO observer gain matrix coefficient
k_2	sfix(16,13)	LO observer gain matrix coefficient
k_3	sfix(16,13)	LO observer gain matrix coefficient
k_4	sfix(16,13)	LO observer gain matrix coefficient

In the following, the Simulink implemented blocks are described in details.

7.4 SineCosine block

Description

The sin(θ_e) cos(θ_e) functions contain the information on the rotor position needed to project the motor model in stator or rotor frame. These trigonometric functions are calculated starting from components of back-EMF in the stator frame.

Arguments

u_{sD}_e_next	direct-axis voltage component in (D,Q) stator frame;
u_{sQ}_e_next	quadrature-axis voltage component in (D,Q) stator frame.
sin(θ_e);	
cos(θ_e).	

Algorithm

Equation 14

$$\cos(\theta_r) = \frac{e_{Q1}}{\sqrt{e_{D1}^2 \div e_{Q1}^2}}$$

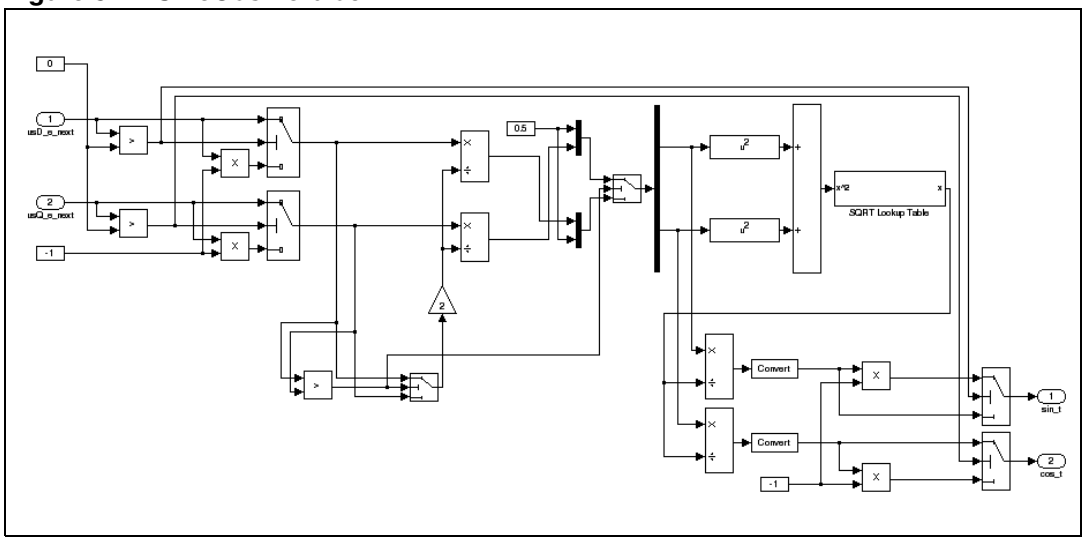
Equation 15

$$\sin(\theta_r) = \frac{e_{D1}}{\sqrt{e_{D1}^2 \div e_{Q1}^2}}$$

Simulink block

The structure of the SineCosine block, in the discrete format, used in the Simulink model is shown in *Figure 5*:

Figure 5. SineCosine block



Test case

In **sinecosine_data.mat** file the inputs and outputs data to test this function are stored.

7.5 LObserver block

Description

This block implements a Luenberger observer where the complete differential motor model (including the electrical and mechanical equations) is arranged to estimate the “state” variables. The back-EMF components in a stator frame are considered the unknown state variables estimated using as feedback the instantaneous error between the estimated and the measured motor currents.

Arguments

isD	direct-axis current component in stator fixed framer;
isQ	quadrature-axis current component in stator fixed framer;
usD_old	direct-axis voltage component in stator frame at the previous step;
usQ_old	quadrature-axis voltage component in stator frame at the previous step;
usD_e_next	direct-axis voltage component in stator frame at the next step;
usQ_e_next	quadrature-axis voltage component in stator frame at the next step.

Algorithm

Equation 16

$$\hat{x}_E = A_E \cdot \hat{x}_E + B_E \cdot u + K \cdot (y - (C_E \cdot \hat{x}_E))$$

Equation 17

$\hat{x}_E = [i_D, i_Q, e_D, e_Q]^T$ vector state variables

Equation 18

$y = [i_D, i_Q]^T$ output vector

Equation 19

$u = [v_D, v_Q]^T$ input vector

K the observer gain matrix

$$A_E = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} \qquad B_E = \begin{bmatrix} B \\ 0 \end{bmatrix} \qquad C_E = \begin{bmatrix} C & 0 \end{bmatrix}$$

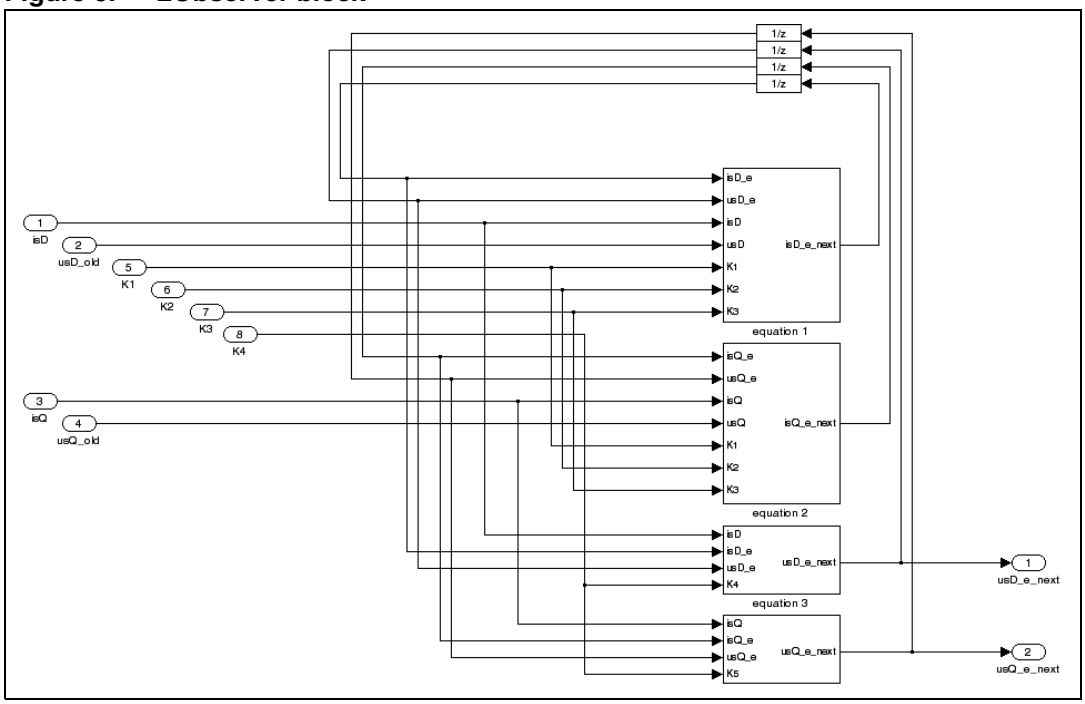
$$A = -\frac{R_s}{L} \qquad B = -\frac{1}{L} \qquad C = I$$

where the superscript ^ denotes the estimated quantities. The system parameters matrices A_E, B_E and C_E are constant (linear system).

Simulink block

The structure of the LObserver block, in the discrete format, used in the Simulink model is shown in *Figure 6*:

Figure 6. LObserver block



Test case

In **Iobserver_data.mat** file the inputs and outputs data to test this function are stored.

8 Software library

8.1 Description

The software library provides the functions for mixed “C” and Assembly programmers on ST10 microcontrollers necessary to implement the rotor position estimation based on a Luenberger observer (LO).

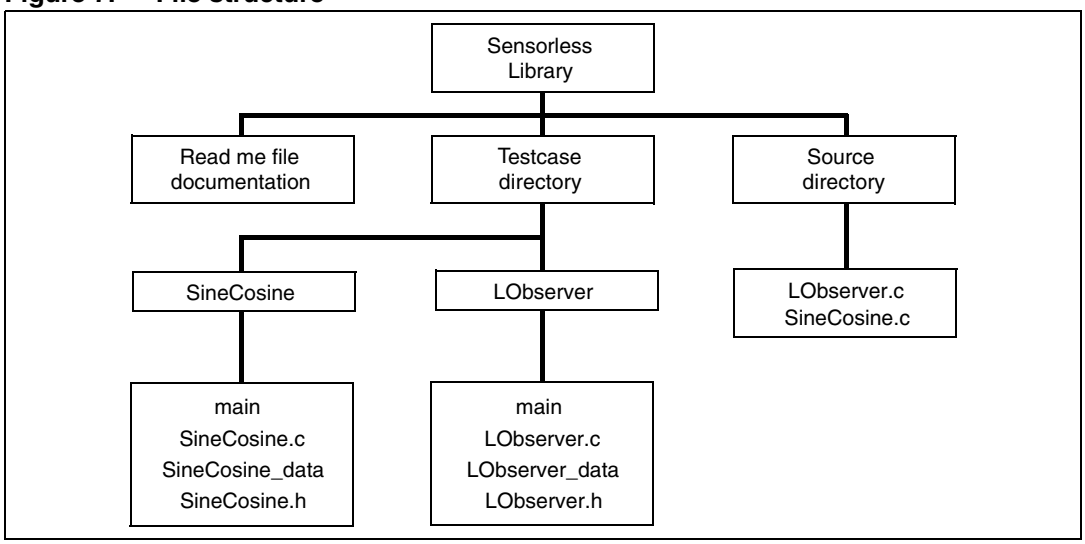
8.2 Using the software library

The 2 main directories of the library package are:

- 1 directory for all test cases: 1 subdirectory per library function.
- 1 directory for all .c sources file: all functions

The folder structure is shown in *Figure 7*

Figure 7. File structure



8.2.1 How to install software library

The software library is delivered as an archive file with .zip extension. To install the software library you need to unzip the file in the directory where you want the library to be copied into.

Note: Please, read the README.txt file in the archive file for specific details on the release.

8.2.2 Tool chain compatibility

The library is compatible with tasking tool chain (V7.5r2 and upwards).

8.2.3 Calling a function

The functions have been written to be called by a C language program.

To include a function in a C language program, you need to:

- include the header file.
- find this .h file in the source directory of the library package.

8.2.4 ST10 MAC configuration

This library has been done for implementing rotor position estimation functions of a 3 phase PMSM (FOC control), using 16-bit data in fixed point precision with different representations (i.e. sfix(16,8), sfix(16,6), etc.). The implemented functions have been optimized with MAC commands using the default configuration (the user have not to change the configuration registers of MAC).

8.2.5 Real time aspects

Any DSP code developed for ST10 can be interrupted at any time and execution resumed after the interrupt routine. There is no added latency when the DSP library is used.

Interrupt routine requirements: the only requirements are only when the DSP unit is used by other tasks that have different priorities: the interrupting task that may interrupt another task using the DSP should save and restore the MAC registers at the entry point and exit point of the routine. (use **#pragma savemac** in Tasking tool chain).

8.2.6 Naming convention

The name of each functions begins with the name of the Simulink equivalent block, that implements it on micro followed by underscore c_step.

8.2.7 Test environment

yyy_data.c: you find the input data vectors and the output data vectors, obtained by Simulink for the same function block, in int16 format.

The name of each test-file begins with the (yyy) function name that it refers, followed by underscore and the suffix “data”.

8.2.8 Flux control library benchmark

The following table gives the characteristics of the main functions of the library:

Table 2. Library capabilities

Function	Code size (bytes)	Nb cycles
Lobserver	313	76
SineCosine	221	114

8.3 Library functions

8.3.1 SineCosine

sinecosine

SineCosine_c_step(int16_T SineCosine_U_usD_e, int16_T SineCosine_U_usQ_e,
int16_T *SineCosine_Y_sin_t, int16_T *SineCosine_Y_cos_t);

Description:

Calculate the funcions sin(θ_e), cos(θ_e) starting from the estimated back-EMF components in the stator frame.

Arguments:

SineCosine_U_usD_e	direct-axis voltage component in stator frame;
SineCosine_U_usQ_e	quadrature-axis voltage component in stator frame;
SineCosine_Y_sin_t	sin(θ_e)
SineCosine_Y_cos_t	cos(θ_e)

Algorithm:

Equation 20

$$\cos(\theta_r) = \frac{e_{Q1}}{\sqrt{e_{D1}^2 \div e_{Q1}^2}}$$

Equation 21

$$\sin(\theta_r) = \frac{e_{D1}}{\sqrt{e_{D1}^2 \div e_{Q1}^2}}$$

Notes:

Test:

To test this function, include the **sinecosine_data.c** file in the current directory.

8.3.2 LObserver

LObserver

```
LObserver_c_step( D_Work_LObserver *LObserver_DWork,
                  ExternalInputs_LObserver *LObserver_U,
                  ExternalOutputs_LObserver *LObserver_Y);
```

Description:

Estimates back-EMF components (state variables) in a stator frame using as feedback the instantaneous error between the estimated motor currents & the measured motor currents.

Data types and structures:

D_Work_LObserver

This structure contains...

```
typedef struct D_Work_LObserver_tag {
int16_T UnitDelayisD_DSTATE;
int16_T UnitDelayisQ_DSTATE;
int16_T UnitDelayusD_DSTATE;
int16_T UnitDelayusQ_DSTATE;
} D_Work_LObserver;
```

ExternalInputs_LObserver

This structure contains the model inputs and the LO observer gain matrix coefficients.

```
typedef struct _ExternalInputs_LObserver_tag {
int16_T isD;
int16_T usD_old;
int16_T isQ;
int16_T usQ_old;
int16_T K1;
int16_T K2;
int16_T K3;
int16_T K4;
} ExternalInputs_LObserver;
```

ExternalOutputs_LObserver

This structure contains the model outputs.

```
typedef struct _ExternalOutputs_LObserver_tag {
int16_T usD_e_next;
int16_T usQ_e_next;
} ExternalOutputs_LObserver;
```

Arguments:

LObserver_DWork	pointer to the..... structure
LObserver_U	pointer to the inputs structure
LObserver_Y	pointer to the outputs structure

Algorithm:

Equation 22 $$\hat{x}_E(k+1) = A_E \cdot \hat{x}_E(k) + B_E \cdot u(k) + K \cdot (y(k) - (C_E \cdot \hat{x}_E(k)))$$

Notes:

Test:

To test this function, include the **lobserver_data.c** file in the current directory. In the .c file you find the inputs and outputs vectors defined as const.

9 Revision history

Table 3. Document revision history

Date	Revision	Changes
02-Apr-2007	1	Initial release.