Nonlinear Controller implementation in a Low-Cost, PC-Based Platform for Induction Motor Drive Research

Technical Themes: I and K

Abstract— **This paper describes the implementation of nonlinear controllers for an induction motor based on exact linearization using a low-cost PC-based platform. The controller algorithm runs in real time in a standard PC fitted with a commercial data-acquisition board and a custom FPGA-based board. The FPGA is used to interface with a PWM inverter and with an incremental encoder fitted in the motor shaft. The controller has been implemented using Real-time Linux as the real-time operating system. The same PC also supports all the control-development tools including compilers, a dynamical-system simulator and a numerical-algorithm package for control design.**

The paper shows experimental results with a 0.25 kW induction motor.

I. Introduction

Nowadays, Digital Signal Processors (DSP) or microcontrollers are frequently used to implement control algorithms for induction motor drives.

On one hand, the most sophisticated DSP's [1] are preferred by the electrical-drive community to implement controllers in research test rigs because they have fast and efficient floating-point-arithmetic units. This characteristics makes it possible to write the algorithms in a high-level language with an almost straightforward implementation. However, these DSP's do not have integrated peripherals such as PWM generators or analog-to-digital converters and, therefore, sophisticated hardware has to be added to the device. In addition, their development systems (hardware and software) require a PC where the DSP board and the peripherals are installed.

On the other hand, microcontrollers (and some special fixed-point DSP's) very often have integrated all the necessary peripherals for electric motor control providing a compact and cheap solution for industrial applications. However, these devices do not have a floating-point-arithmetic unit and the algorithms have to be written using fixed-point arithmetic. Programming and debugging can still be done in a high-level language but an additional PC is also necessary for that purpose.

In this paper, an alternative approach is presented to implement a research test rig where to test complicated nonlinear controllers for induction motors: the same PC is used for controller development, programming and debugging (possibly using simulation) and for controller real-time implementation. Given the fact that today's microprocessors used in PC's (Pentium or Athlon) are very powerful, this was the platform chosen to investigate the implementation of complex nonlinear controllers based on exact linearization for an induction motor drive where high sampling frequency was required. The controllers required state-variable and parameter identification [2].

The alternative proposed here required to fit the PC with some external peripherals. A typical data-acquisition board was used for current feedback while a custom FPGA-based board was used for the final stage (timing) of vector PWM implementation and speed feedback using an incremental encoder. Everything together made a cost effective, reliable and easy to maintain research platform.

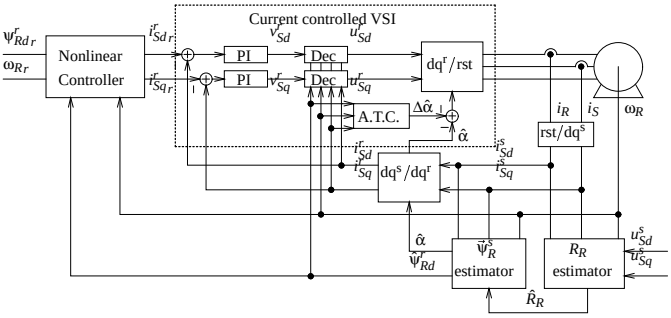

Fig. 1. Controller implementation block diagram.

The paper is structured as follows. A brief explanation of the controller implemented in the test rig is given in section II. The test rig used is presented in section III. The discussion includes the custom hardware designed to interface with the inverter and the incremental encoder, and the real-time implementation of the control algorithm. Finally, the experimental results obtained are presented in Section IV. The main conclusions of the paper are summarised in Section V.

II. Controller implementation

A Lyapunov based nonlinear controller and a conventional direct vector controller were implemented in a research test rig. The controllers are described in [2] where it is shown that both controllers are some form of exact linearization using feedback. Only a brief description will be done in this paper to illustrate the computational complexity of the implemented controllers.

A block diagram of the controller is shown in Figure 1. The controller measures the rotor speed and two motor currents with hall-effect sensors. The stator-current components in stator coordinates are calculated in the block "rst/dq^s" using Park's transformation (see [3]) from the two phase currents. Stator-current components and stator-voltage components[1] are used for rotor-resistance and rotor-flux estimation. The rotor-flux estimator is an open-loop reduced order one with the following discrete-time implementation:

[1]Stator-voltage is not measured but taken directly from the current-controller output.

$$\left(\hat{\vec{\psi}}_R^s\right)_k = e^{\vec{a}T_s}\left(\hat{\vec{\psi}}_R^s\right)_{k-1} + \frac{\vec{b}}{\vec{a}}\left(e^{\vec{a}T_s} + \frac{1}{\vec{a}T_s}\left(1-e^{\vec{a}T_s}\right)\right)\left(\vec{i}_S^s\right)_{k-1} - \frac{\vec{b}}{\vec{a}}\left(1+\frac{1}{\vec{a}T_s}\left(1-e^{\vec{a}T_s}\right)\right)\left(\vec{i}_S^s\right)_k \quad (1)$$

Where:

$$e^{\vec{a}T_s} = e^{ET_s}\left(\cos(\omega_R T_s) + j\sin(\omega_R T_s)\right) \quad (2)$$

$$\vec{a} = E + j\omega_R \quad (3)$$

$$\vec{b} = D \quad (4)$$

In these equations, E and D are functions of the motor parameters, ω_R is the rotor speed and T_s is the sampling period.

Given the open-loop nature of the rotor-flux estimator, it is necessary to estimate the rotor resistance of the machine to avoid flux-estimation errors. The estimator presented in [4] has been used. This estimator is based in a lineal regression model of the motor:

$$\overbrace{\vec{v}_1 - j\omega_R\int\vec{v}_1 dt}^{\vec{y}} = R_R\overbrace{\left(\frac{-1}{L_R}\int\vec{v}_2 dt\right)}^{\vec{u}} \quad (5)$$

where:

$$\vec{v}_1 = \vec{u}_S^s - R_S\vec{i}_S^s - \sigma L_S\frac{d}{dt}\vec{i}_S^s \quad (6)$$

$$\vec{v}_2 = \vec{u}_S^s - R_S\vec{i}_S^s - L_S\frac{d}{dt}\vec{i}_S^s \quad (7)$$

The value of R_R can be computed as the quotient between $\vec{u}$ and $\vec{y}$. The derivatives and integrals of equations (6) and (7) are computed using a fourth-order, band-pass Bessel filter. The values of $\vec{u}$ and $\vec{y}$ are averaged using a recursive total-least-squares algorithm to avoid the effects of noise. It is worth to mention that this algorithm cannot be used when the machine is under fast speed transients. Therefore, the rotor-resistance estimation is only computed while the motor is at steady state.

The rotor-flux estimator gives the magnitude ($\hat{\psi}_{Rd}^r$) and angle ($\hat{\alpha}$) of the rotor-flux vector. The former is used for flux closed-loop control while the latter is used to calculate the d-q stator-current components in field coordinates (block "dq^s/dq^r" in Figure 1).[2] These currents are used in the microprocessor to control the inverter with third order linear PI controllers. To avoid using a high-gain, high-speed current controllers, these controllers are designed taking into account the stator dynamics, which gives better results when low sampling frequencies are used [2]. Unfortunately this approach shows coupling between the d and q current components. Decoupling is achieved in "Dec" block of Figure 1, implementing a variable change given by:

$$u_{Sd}^r = v_{Sd}^r - \frac{\omega_S i_{Sq}^r + B\psi_{Rd}^r}{F} \quad (8)$$

$$u_{Sq}^r = v_{Sq}^r + \frac{\omega_S i_{Sd}^r + C\omega_R\psi_{Rd}^r}{F} \quad (9)$$

[2]This is really the calculation of the current components in a reference frame rotated an angle $\hat{\alpha}$ with respect to the stator reference frame.

	Rotor-flux controller
Vec.	$i_{Sdr}^r = K_{I\Psi}\int\left(\psi_{Rdr}^r - \psi_{Rd}^r\right)dt + K_{P\Psi}\left(\psi_{Rdr}^r - \psi_{Rd}^r\right)$
L-I	$i_{Sdr}^r = \frac{1}{D}\left[k_1 I_1\int\left(\psi_{Rdr}^r - \psi_{Rd}^r\right)dt + (k_1+I_1)\left(\psi_{Rdr}^r - \psi_{Rd}^r\right) - E\psi_{Rd}^r\right]$
	Rotor-speed controller
Vec.	$i_{Sqr}^r = \frac{1}{G\psi_{Rd}^r}\left(K_{I\omega}\int(\omega_{Rr}-\omega_R)dt + K_{P\omega}(\omega_{Rr}-\omega_R)\right)$
L-I	$i_{Sqr}^r = \frac{2J}{PG\psi_{Rd}^r}\left[k_2 I_2\int(\omega_{Rr}-\omega_R)dt + (k_2+I_2)(\omega_{Rr}-\omega_R)\right]$

TABLE I
Lyapunov-Integral (L-I) and Direct Vector (Vec) Controllers equations.

where v_{Sd}^r and v_{Sq}^r are the outputs of the decoupled current controllers, u_{Sd}^r and u_{Sq}^r are the stator-voltage vector components in rotor-flux coordinates, B, C and F are functions of the motor parameters and ω_S is the rotor-flux-frame speed, which is computed as:

$$\omega_S = \omega_R + \frac{D\,i_{Sq}^r}{\psi_{Rd}^r} \quad (10)$$

The decoupled stator-voltage vector is fed into the block "dq^r/rst" where it is transformed from rotor-flux coordinates to stator coordinates before computing the switching times for the inverter. The angle used for the transformation is $\hat{\alpha} + \omega_S\frac{T_s}{2}$. The latter is calculated in block "A.T.C" (Axis Turn Compensation) to compensate the rotation of the rotor-flux-oriented frame during the sampling period (see [5]). This compensation is vital at high rotor speed.

The flux magnitude and the rotor speed are also fed into the "Nonlinear Controller" block, which implements the rotor flux and rotor speed controllers. The outputs of these controllers are the reference currents which are fed into the PI current controllers. A nonlinear, Lyapunov-based controller has been tested, comparing his performance against a conventional direct vector controller. The equations used in each controller are shown in table I.[3]

III. Test-rig setup

A block diagram of the test rig developed is shown in Figure 2. The test rig has a 250 W induction machine with an incremental encoder for speed measurement. A d.c. machine has been used to load the induction motor. The field winding of this machine is fed by a d.c. voltage source and the armature winding is connected to a variable resistor. With this setup the load torque is always proportional to the rotational speed of the machine. The control system has been implemented in a PC compatible based in a 200 MHz Pentium processor. This PC has been fitted with a data-acquisition board from Keithley Metrabyte (DAS-1400) which is used to measure the currents thorough two Hall-effect current sensors. A FPGA-based card has been designed to trigger the inverter legs and to measure the encoder pulses. The real-time control software has been implemented using the RT-Linux operating system [6]. The controller sampling frequency has been chosen equal to 2 kHz.

[3]In the full paper, these equations will be explained in more detail.

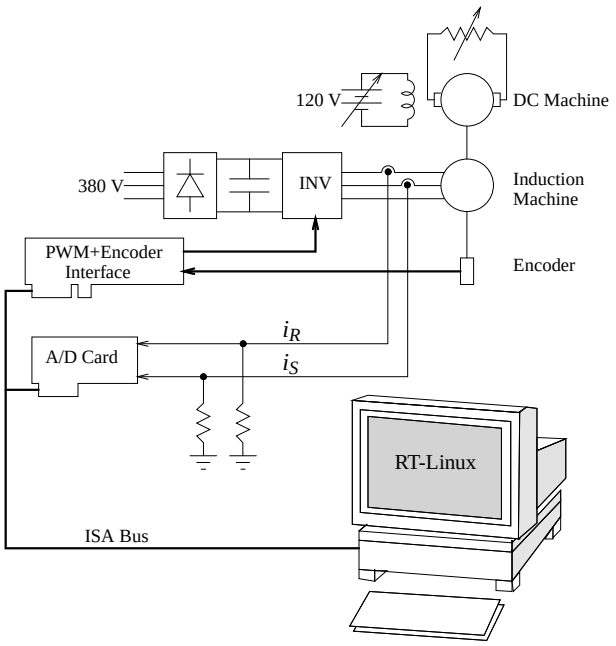

Fig. 2. Test Rig block diagram.

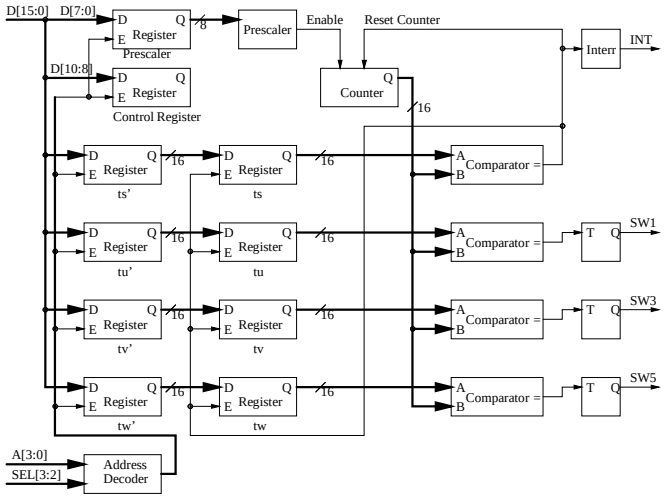

Fig. 3. PWM generator block diagram.

A. Inverter control

The inverter is controlled by space-vector modulation [7], which makes it possible to place the voltage vector in any position of the d-q plane. However, to make this placement accurate, it is necessary to control the switching time of the three inverter legs. It is not feasible, given the typical latency of microprocessors of various microseconds, to rely in an interrupt routine to look after the inverter switching. In consequence, it is necessary to use dedicated hardware to control the switching times of the inverter legs. This hardware can be:

- Integrated in the microprocessor die.
- Built using discrete LSI timers.
- Configured in a Programmable Logic Device.

The first option is frequently found in modern 16 bits fixed point microcontrollers and DSP's.[4] Unfortunately, the development platforms of these microprocessors can be very expensive and flexible floating-point computations becomes complicated. In addition, the fact that the user base of these tools is not very large makes these systems bug-prone. This work has investigated the application of a general purpose Pentium microprocessor, which is powerful enough to cope with the control+estimator algorithm in floating-point arithmetic and have economy-scale prices in both, the microprocessor and the development tools. With this approach, it has been necessary to use specific hardware to generate the PWM signals. Using programmable logic has provided more flexibility than discrete components.

A.1 PWM generator block diagram

A block diagram of the PWM generator circuit is shown in Figure 3. The circuit has a 16 bit counter and 4 comparators associated to 4 registers of 16 bits. These registers store the sampling period t_s and the switching times of each inverter leg: t_u, t_v and t_w. When the counter is equal to any of these times, the pulse generated at the output of the comparator toggles the inverter driver output. When the counter reaches the value stored in the t_s register, an interrupt is generated which acts as a time-base for the controller program. This signal is also used to transfer the values stored by the microprocessor in the registers t_s', t_u', t_v' and t_w' to the t_s, t_u, t_v and t_w. This double buffer technique allows the processor to write the switching times at any time during the previous switching period. However, this results in a delay of one sampling period in the application of the desired voltage vector. This delay has to be compensated in the discrete time implementation of the controllers.

[4]Examples of these microprocessors are the C166 from Infi neon Technologies [8] or the TMS320F240 from Texas Instruments [9].

B. Incremental-encoder interface

With the exception of the sensorless control algorithms, the rotor position or speed has to be measured. Also in the sensorless drives, the rotor speed and position has to be measured in the test rig to check the controller performance. Today the most common device used to speed and position measurement is an incremental encoder. The necessary circuitry to interface with the encoder has been implemented in the same FPGA used for PWM generation. A block diagram of the encoder interface circuit is shown in figure 4. The circuit has two inputs that will be connected to the two channels of the encoder. These signals are filtered to eliminate noise spikes that could be taken as encoder pulses. The filtered signals are feed into the "Rotation Direction" block to get the rotation direction of the encoder axis and to the "x 4" block, where a pulse is generated at the output for each edge of the inputs, multiplying by four the number of pulses counted in the 16 bit counter block. The outputs of the circuit are the pulse count and the rotation direction, which can be read by the microprocessor to compute the rotor speed and angular position.

C. Real-time implementation of the control algorithm

The program to implement the control algorithm must be executed each sampling period, independently of the computer

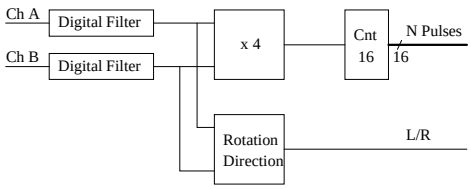

Fig. 4. Encoder interface block diagram.

load. Therefore, it is necessary to use real-time-programming techniques to implement this control program. There are two alternatives:

- Custom programming. If the program is implemented in an embedded system using a microcontroller or a DSP, it is possible to implement it directly using real-time programming techniques [10]. In this case the common approach consist in an endless loop in which the computer waits the beginning of the sampling period and then measures the inputs and computes the outputs based on the control algorithm.
- Real-Time operating system. If, in addition to the control algorithm, it is necessary to implement other tasks in the control microprocessor, like data-logging or an user-interface, the custom programming approach can be very difficult. In this case using an operating system facilitates the implementation of the controller. However, time-sharing operating systems like Linux or Windows-NT are designed to offer the best performance on average, but not to guarantee a maximum response time.[5] This fact makes it necessary to use a Real-Time Operating System (RTOS) when the sampling time is less than 1 s. RTOS such as QNX or VxWorks are feature rich and have a low memory footprint but they are very expensive. In this application an open-source RTOS (RT-Linux) has been chosen instead. This RTOS is based in a simple real-time scheduler under which the Linux O.S. run as the lower priority task. With this approach only the time-critical task (the non-linear controller in this application) run in Real-Time space. The rest of the controller tasks like the user interface and data-logging are Linux applications that run in user space, isolated from the real-time tasks. The communication between real-time task and user space tasks is done over FIFO devices and shared memory.

IV. Experimental results

Figure 5 shows the results obtained with the controller described in section II implemented in the test rig described in section III. The data are taken from the controller task and sent to the supervision task using a RT-FIFO. The supervision task stores the data into a disk file to further analysis with computer packages like Matlab or Octave. In this case the figure has been plotted using Octave and GnuPlot.

V. Conclusions

This paper explains the implementation of some nonlinear-control algorithms for induction motor drives in a research test rig using standard PC hardware instead of DSP's or microcontrollers. It has been shown that with the addition of the acquisition and modulation hardware and the installation of a real-time operating system, a robust and easy to use platform is obtained.

While it is easy to get a data acquisition card to measure current and voltages and an interface card with an incremental encoder in the market, it is difficult to get a tree-phase modulation card. This has motivated the design of a custom card, based in a FPGA, to perform the timing required to control the switching times of the inverter legs.

The controller, state-variable estimators and parameter estimators have been implemented in a standard PC using the Real-Time Operating System RT-Linux. The interface with the inverter and the incremental encoder has been implemented in a custom board based on a FPGA. This approach has resulted in a very flexible and low-cost environment.

Experimental results have been shown to illustrate the feasibility and performance of this approach.

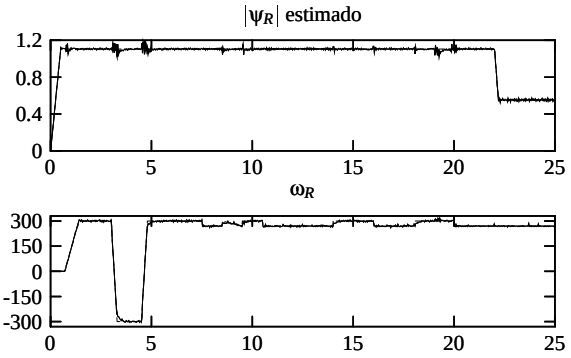

Fig. 5. Results of a Lyapunov controller in rotor-flux coordinates. Sampling freq. 2kHz. X axis in seconds. Y axis in Wb (top) and $\mathrm{rad_{el}/s}$ (bottom).

References

[1] Texas Instruments Inc., *TMS320C6000 CPU and Instruction Set Reference Guide*, October 2000.

[2] José Daniel Muñoz Frías and Aurelio García Cerrada, "A comparative study between two nonlinear control techniques for induction motor drives," in *IECON02*, Sevilla, 2002.

[3] Paul C. Krause, *Analysis of Electric Machinery*, McGraw-Hill Inc., New York, 1986.

[4] A. García-Cerrada and J.L. Zamora, "On-line rotor-resistance estimation for induction motors," in *Proc. of the European Power Electronics Conf. (EPE'97)*. EPE, September, Trondhein 1997, vol. 1, pp. 542–547.

[5] Aurelio García Cerrada, *Observer-based field-oriented controller for an inverter-fed traction induction motor drive*, Ph.D. thesis, University of Birmingham (U.K.), July 1991.

[6] Michael Barabanov, "A linux-based real-time operating system," M.S. thesis, New Mexico Institute of Mining and Technology, Socorro, New Mexico, June 1997.

[7] Joachim Holtz, "Pulsewidth modulation for electronic power conversion," *Proceedings of the IEEE*, vol. 82, no. 8, pp. 1194–1214, August 1994.

[8] Infineon Technologies, *C166 Family of High-Performance CMOS 16-bit Microcontrollers*, Infineon Technologies AG, München, 1999-08 edition, 1999.

[9] Texas Instruments, *TMS320C240, TMS320F240 DSP controllers data sheet*, Texas Instruments, Houston, Texas, 1998.

[10] David M. Auslander, John R. Ridgely, and Jason C. Jones, "Real-time software for implementation of feedback control," in *The Control Handbook*. CRC Press and IEEE Press, 1996.

[11] Michael Barabanov and Victor Yodaiken, "Introducing real-time linux," *Linux Journal*, , no. 34, February 1997.

[5] For example, in a Pentium 200 microprocessor running Linux, the average interrupt response time is 2μs. However the maximum response time is not limited, and it can be hundred of milliseconds when the machine is heavy loaded [11].