ELSEVIER

Microprocessors and Microsystems 22 (1999) 523–534

MICROPROCESSORS AND MICROSYSTEMS

Fuzzy logic speed control of an induction motor

Jaime Fonseca*, João L. Afonso, Júlio S. Martins, Carlos Couto

Department of Industrial Electronics, Minho University, Largo do Paço, 4709-Braga Codex, Portugal

Received 30 January 1998; received in revised form 31 August 1998; accepted 7 October 1998

Abstract

This paper describes the use of fuzzy logic techniques to control the speed *of a three-phase induction motor*. The use of *Matlab/Simulink* and fuzzyTECH MCU96 as software development tools for system design is emphasised. Hardware implementation is based on a standard 16/32-bit microcontroller, without the need of any additional components for the fuzzy logic controller. The system performance is evaluated in comparison with a traditional PI control scheme. Both simulation and experimental results are presented. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Real-time control; Fuzzy logic; Induction motor; Slip control

1. Introduction

Fuzzy logic control is one of the most interesting fields where fuzzy theory can be effectively applied. Fuzzy logic techniques attempt to imitate human thought processes in technical environments. In doing so, the fuzzy logic approach allows the designer to handle efficiently very complex closed-loop control problems, reducing, in many cases, engineering time and costs [1,2]. Fuzzy control also supports nonlinear design techniques that are now being exploited in motor control applications [3,4,5]. An example includes the ability to distribute gain over a range of inputs in order to avoid the saturation of the control capability. Software based implementations of these techniques have been mainly used in industrial automation for relatively slow processes. Fast fuzzy control usually requires the use of a specific hardware processor.

Initially fuzzy control was found particularly useful to solve nonlinear control problems or when the plant model is unknown or difficult to build. In this article it will be shown that these techniques can also be useful in applications where classical control performs well. Fuzzy Logic allows a simpler and more robust control solution whose performance can only be matched by a classical controller with adaptive characteristics, much more difficult to implement.

This article reports work that is being done in order to apply fuzzy techniques to control the speed of an induction motor. A conventional PI slip controller, implemented with a standard microcontroller, was used as a reference and as a starting point. The goal here is, by simple software modification, to replace the PI scheme by a fuzzy controller and try to improve its performance. Attention will be focused on the use of software development tools that support fuzzy and neurofuzzy design (fuzzyTECH), and simulation (Matlab/Simulink), rather than on the details of induction motor control.

From the application of fuzzy control arises two problems: how to select the fuzzy control rules and how to set the membership functions. Two approaches are normally used to accomplish this task. One consists of acquiring knowledge directly from skilled operators and translate it into fuzzy rules. This process, however, can be difficult to implement and time-consuming. It happens quite often that the operator is not able to express his knowledge of the process explicitly and accurately. As an alternative fuzzy rules can be obtained through machine learning techniques, where the knowledge of the process is automatically extracted or induced from sample cases or examples. Many machine learning methods developed for building classical crisp logic systems can be extended to learn fuzzy rules. A recently very popular machine learning method is Artificial Neural Networks (ANN), which have been developed to mimic biological neural systems in performing learning control and pattern recognition. Another machine learning method is the Genetic Algorithms (GA) approach introduced by Holland [6]. Genetic algorithms are adaptive and probabilistic search algorithms, based on natural selection and natural genetics.

The use of these self-learning techniques were tried for

* Corresponding author. Tel.: 51 53 615010; fax: 351 53 615046.

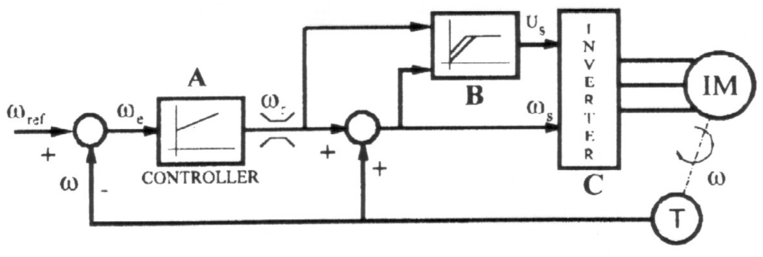


Fig. 1. Slip control scheme for an induction motor voltage-source inverter drive.

this case but the results were not yet very convincing, and for that reason will not be covered in this contribution. Because this is a well-known control case the fuzzy control rules and the membership functions were selected based on empirical methods.

This article is organised as follows: Section 2 introduces the induction motor slip control scheme, Section 3 describes the use of development tools in system design, in Section 4 simulation results are presented, while experimental results are reported in Section 5, with conclusions in Section 6.

2. The induction motor speed control

2.1. Slip control

With the exception of high performance drives, induction motor speed can be controlled through a simple slip control scheme like the one presented in Fig. 1 [7,8]. The machine is fed by a voltage-source pulse width modulated (PWM) inverter (block C), which is a power converter that produces three phase AC voltages that can be varied both in frequency and amplitude. Rotor speed ω is measured using a digital tachometer (T) and compared with speed reference ω_{ref}.

Assuming that fast response is not required, a linear approximation of the induction motor steady state model can be used [9]. The motor output torque is almost proportional to the slip frequency ω_r, which is the difference between the motor supply frequency ω_s and motor speed. However, slip frequency must be limited so that the motor model is valid. In doing so, a limit to both peak torque and stator current is indirectly set.

In this conventional control scheme, the speed error ($\omega_e = \omega_{ref} - \omega$) is input to a PI controller (block A) which sets the motor slip frequency. PID controllers are usually avoided because of the noise generated by the commutation of the inverter switches. Stator frequency is obtained by adding the slip frequency to rotor speed, and stator voltage (U_s) is set accordingly to a predefined $U_s/\omega_s \approx$ constant law (block B), so that motor flux is kept at its nominal value. Voltage and frequency values are then input to the voltage source inverter.

This article presents an alternative to the implementation of the controller block A using fuzzy logic. Both approaches are detailed and evaluated in the following pages.

2.2. The drive arrangement

A 1 kW three-phase induction motor fed by a voltage source type inverter was used in the experiments. The inverter was implemented using a 6 IGBT power module, A standard Intel 80C196KD microcontroller performs the control algorithm and generates the PWM waveforms for the IGBT motor drive inverter.

The system control hardware is based on the 196KD

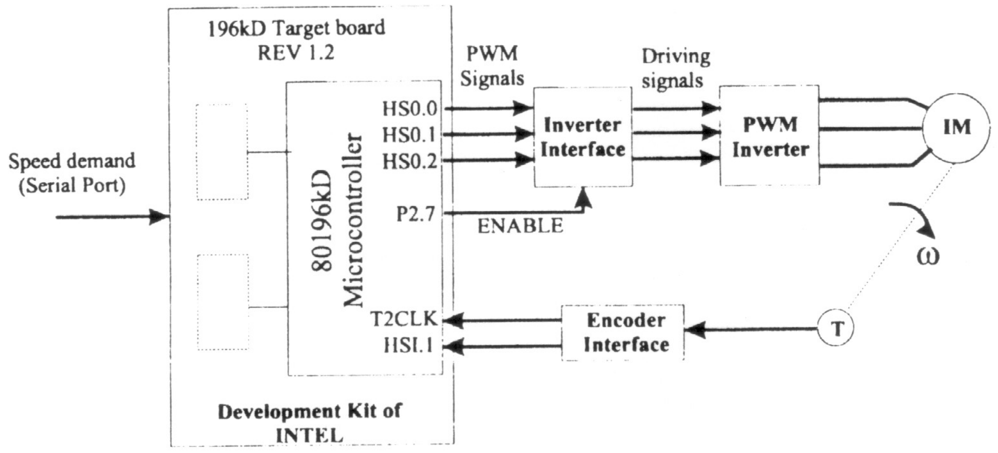


Fig. 2. Simplified diagram of the hardware implementation.

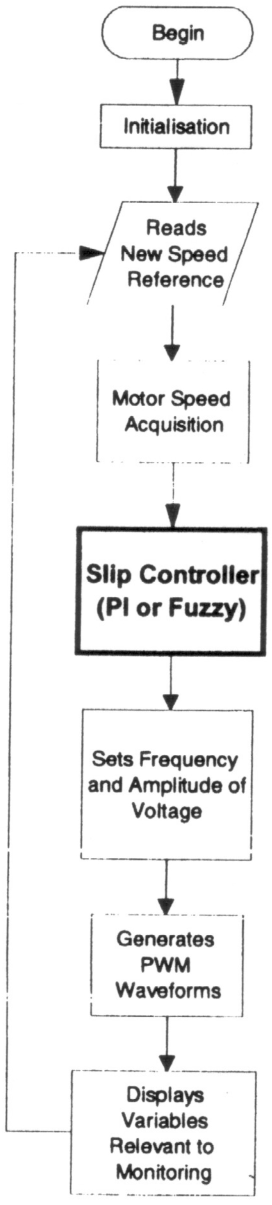

Begin
Initialisation
Reads New Speed Reference
Motor Speed Acquisition
Slip Controller (PI or Fuzzy)
Sets Frequency and Amplitude of Voltage
Generates PWM Waveforms
Displays Variables Relevant to Monitoring

Fig. 3. Drive flow chart.

Target Board Rev1.2. It is a simple development kit from INTEL, which includes a 20 MHz 80C196KD microcontroller, 32 kB of RAM, a ROM for a small monitor, and a serial interface. No fuzzy processors or any other specific hardware was used to implement the fuzzy logic controller. As shown in Fig. 2 very little extra circuitry was added to the Target Board: just two circuits, one to interface with the IGBT power inverter (*Inverter Interface*) and another to interface with the digital tachometer (*Encoder Interface*). The Inverter Interface circuit receives PWM signals (TTL levels) from the microcontroller and produces compatible driving signals to the PWM Inverter (IGBT power module). This circuit is also responsible for adding dead times to the driving signals, to avoid short-circuits through IGBTs of a same branch of the PWM inverter.

The Encoder Interface circuit receives the tachometer signals and makes them suitable for the microcontroller digital inputs.

A terminal can be connected to the system through the microcontroller serial port to set the speed reference and to monitor some of the control variables.

2.3. Software structure

The software structure can be described according to the flow chart presented in Fig. 3. After the initialisation of some internal variables, there is an endless loop where the speed reference is read and the motor speed is acquired from the tachometer. The speed error (the difference between reference speed and motor speed) is then computed and used as input to the slip controller (fuzzy logic or PI controller). The controller sets the motor slip frequency which allows the amplitude and frequency voltage to be computed according to the block diagram in Fig.1. These values are input to the software module that synthesises the PWM waveforms. Finally, some relevant control variables are displayed for monitoring purposes.

The main objective of this article is the slip controller block implementation, which will be discussed in the next sections. The code for this block was automatically generated by fuzzyTECH (for the case of the fuzzy controller). This code includes the fuzzification of the input variables, the inference process (execution of the activated rules), and the defuzzification process (production of the output variable). The PI controller was implemented using C language.

The main control loop, which performs the data acquisition, the fuzzy controller and the PWM waveform generation, runs within 5.1 ms sampling time in the 80C196KD microcontroller with a 20 MHz clock. The fuzzy slip controller execution time for the same processor is 2.5 ms.

All the other software blocks will not be detailed here. They were also developed by the authors, mainly in C language, with some fast routines written in assembly language.

3. Control system design

3.1. Development tools

3.1.1. Matlab/Simulink

Matlab is a high-level language oriented toward engineering and scientific applications. It has evolved over a ten-year history to become a popular, flexible, powerful, yet simple language. It has served as an effective platform for more than twenty toolboxes supporting specialised engineering and scientific applications, covering areas from symbolic computation to digital filter design, control theory, fuzzy logic, and neural nets. It is to be used interactively, and supports also the ability to define functions and scripts, and dynamically links with C and Fortran programs. Recent trends in the Matlab language have focused on an object

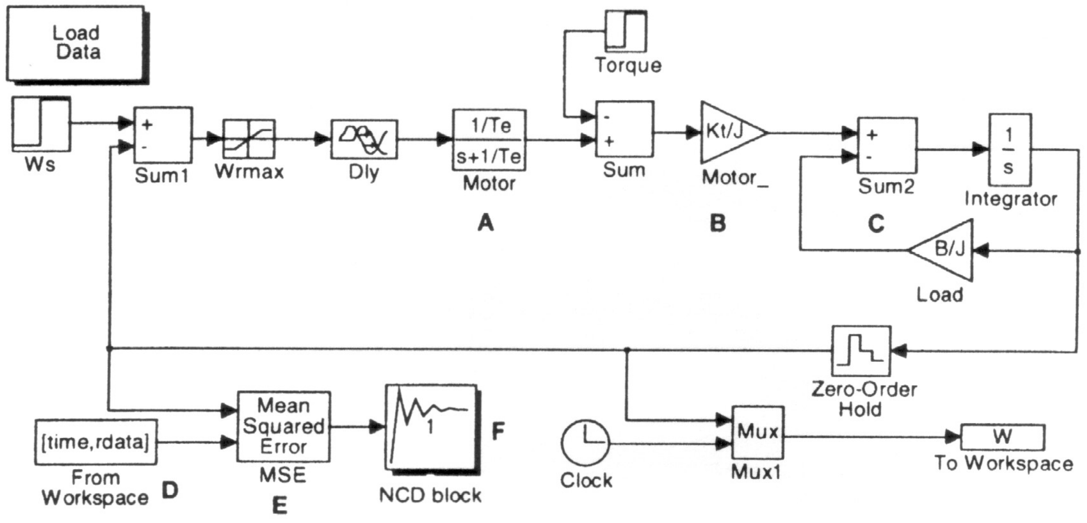


Fig. 4. *Simulink* block diagram for motor model identification.

oriented graphics capability that permits a rich Graphical User Interface (GUI) construction [10,11].

Simulink is built on top of the Matlab. It is an interactive environment for modelling and simulating a wide variety of dynamic systems, including linear, nonlinear, discrete-time, continuous-time, and hybrid systems. It combines the power and ease-of-use of an application package with the flexibility and extensibility of a language. The user can build block diagram models with click-and-drag operations, change model parameters on-the-fly, and display results ''live'' during a simulation. This tool is also a uniquely open system, allowing to choose, adapt, and create software and hardware components to suit each application [10,12].

Together, Simulink and Matlab provide an ideal integrated environment for developing models, performing dynamic system simulations, and designing and testing new ideas.

3.1.2. The nonlinear control design toolbox

The *Nonlinear Control Design* (NCD) toolbox operates with Matlab and Simulink to provide a powerful time-domain-based optimisation technique for designing linear and nonlinear control systems [13].

Controller designs are developed as block diagram models in Simulink. Constraints are specified interactively using a powerful graphical interface. As a result, attaining performance objectives and optimising tuneable parameters becomes an easy and intuitive process. The NCD toolbox automatically converts time-domain constraints and time responses into an optimisation problem, and then solves the problem using state-of-the-art algorithms, thereby tuning the selected model parameters.

Many of the automatic control design methods commonly used today (e.g., LQG/LTR and H∞) require that the plant and controller be approximated by linear models. In contrast, the NCD toolbox uses a new approach to computer-aided control system design (CACSD) that allows you to directly tune parameters in nonlinear systems. Plants and controllers can be continuous-time, discrete-time, or hybrid. Controllers can take any form, including PID, state feedback, observer-based, or decentralised. The main features of the NCD toolbox can be summarised as follows:

- Any plant or control structure modelled with Simulink can be used, including linear/nonlinear, continuous/discrete/hybrid, single-input/single-output (SISO), or multi-input/multi-output (MIMO).
- Operation is fully automated, so there is no need to write additional code.
- Constraints can be placed on any output signals, including actuators, command tracking outputs, and disturbance measurements.
- Monte Carlo methods can be used to increase system robustness for problems with plant uncertainty.
- The optimization solver can be stopped at any time, thereby enabling access to intermediate solutions.
- There are no limits on the number of tuneable parameters, which can be scalars, vectors, or matrices.
- An automatic overachieving option attempts to adjust the response so that it passes through the midpoint of the constraints.

3.1.3. The fuzzyTECH MCU-96 Edition

The fuzzyTECH MCU-96 Edition was used to design the fuzzy logic controller. It is a full graphical tool that supports all design steps for fuzzy system engineering: structure design, linguistic variables, rules definition, and interactive debugging. Moreover, this tool generates C-code with optimised assembly functions to the Intel MCS-96 microcontrollers family and others [14,15]. It also produces Mcode, which can be used for system representation in simulation and mathematical software packages.

Within the fuzzyTECH tool it is possible to use the *NeuroFuzzy* Module which allows the automatic generation

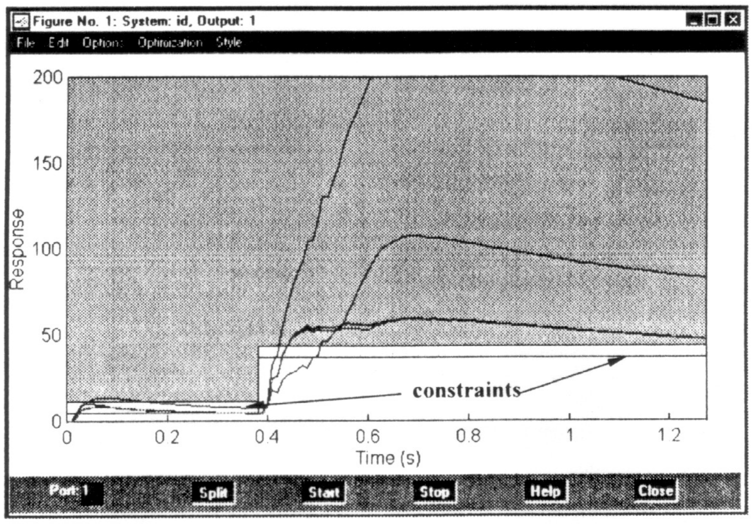


Fig. 5. NCD constraint editor dialog box.

of membership functions and rules, as well as their optimisation towards the data sets. It uses a modified error back propagation algorithm to train the rules and the membership functions of a fuzzy logic system.

The user does not need to worry about the details of the algorithm because these tools work as an intelligent assistant to the design, helping to generate and optimise membership functions and rule bases from sample data.

3.2. Model Identification using the NCD toolbox

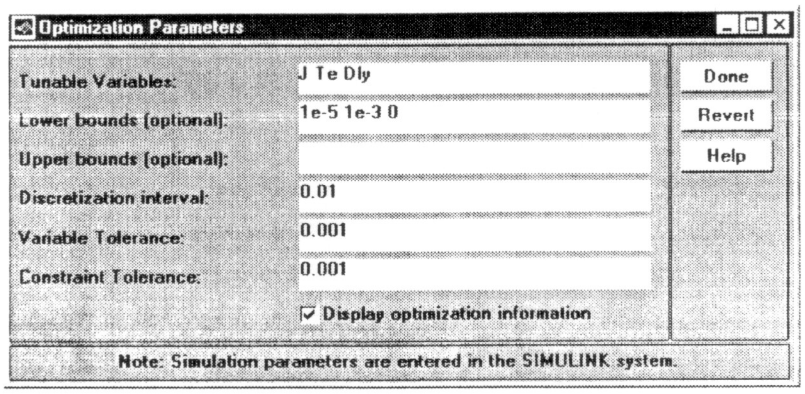
The induction motor model was obtained, as a first approximation, using standard tests (no-load, DC, and lockedrotor tests). The NCD package was then used as an identification tool to fine-tune the motor parameters obtained experimentally. The basic idea is to compare experimental data obtained from the motor open-loop speed response to a step in stator frequency, with the simulated signal assuming the same input and the motor model with the parameters obtained experimentally (Fig. 4). If the parameters are correct then both signals should agree.

The usual approach to perform system identification involves constraining some function of error signals. In this example, the ''Mean Squared Error'' block (E) is used to compare (subtract) the experimental and simulated data. The ''From Workspace'' Simulink block (D) is used to import the observed motor speed data into the system.

The error signal, as defined above, will always be positive, and the goal of the NCD optimisation algorithms is to drive it as close to zero as possible (it should always be zero if the motor parameters were correct). For that purpose constraints are adjusted with the help of the ''NCD Constraint Editor dialog box'', as depicted in Fig. 5. Constraints are set to zero before the input step in stator frequency (at 0.4 s), and to a value close to zero after that.

The parameters, which are tuneable for optimisation, must be defined before starting the optimisation. This can be done through the ''Optimisation Parameters Dialog Box'', as shown in Fig. 6.

The parameters adjusted in this case were the motor electrical time constant (T_e), the motor-load inertia (J) and the system delay (Dly). The initial values for these parameters were obtained experimentally.



Tunable Variables:	J Te Dly
Lower bounds (optional):	1e-5 1e-3 0
Upper bounds (optional):	
Discretization interval:	0.01
Variable Tolerance:	0.001
Constraint Tolerance:	0.001

☑ Display optimization information

Note: Simulation parameters are entered in the SIMULINK system.

Fig. 6. Optimisation parameters dialog box.

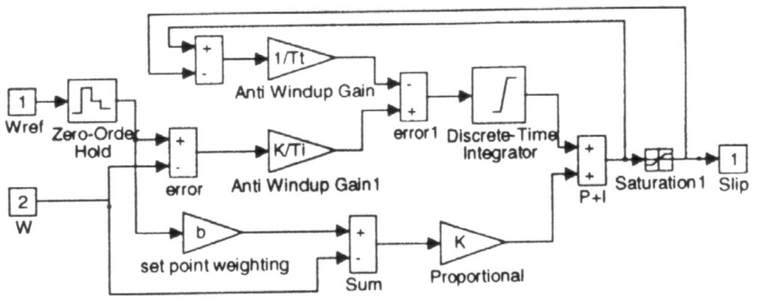


Fig. 9. PI Controller block diagram expanded.

The evolution of the optimisation process in terms of the error signal can be followed in Fig. 5. At the beginning the signal is far from zero (the first curve is even out of range which is irrelevant in this case) but it approaches minimum values (within the specified constraints) as the optimisation process is carried out and the motor parameters are tuned to its optimal values.

The merits of using the NCD toolbox are obvious when the measured and the simulated motor speed responses are compared, before and after the parameter identification process (Fig. 7a,b): at the end both signals are almost coincident.

The NCD toolbox is very easy to use and performs quite well, but like any optimisation tool requires a bit of strategy. It is not a good idea, for instance, to try to adjust all the parameters at the same time, or to put too much ''pressure'' on the constraints all the way from the very beginning of the optimisation procedure.

3.3. Controller Design

In order to evaluate the merits of the fuzzy logic techniques compared to a classical approach, the induction motor slip controller of Fig. 1 was implemented first using PI control, and then fuzzy logic.

3.3.1. Optimising PI controller parameters using the NCD toolbox

Fig. 8 presents the system block diagram used to perform the PI controller simulations in the Simulink environment. The ''Zero-Order Hold block'' is used to set the simulation sampling time equal to 5.1 ms, which is the value used in the practical implementation. Note that because of slip limitation, which introduces a nonlinearity at the controller output, a PI with an anti-windup mechanism (Fig. 9) must be used [16].

The PI controller was first designed through a classical control approach (root-locus). Then the NCD toolbox was used to optimise its response to a speed reference step and to minimise the speed variation when a torque disturbance is applied. The procedure was identical to the one described for the motor model identification. The main difference is that there is no need to produce error signals here: constraints are just set to shape the desired speed response in terms of rise, fall and settling times, and overshoot values (Fig. 10). The tuneable parameters were the proportional gain (K), the integral gain ($1/T_I$), and the antiwindup gain ($1/T_t$).

3.3.2. The fuzzy logic approach

The fuzzyTECH MCU-96 Edition used to develop this work covers the following steps of a fuzzy logic design:

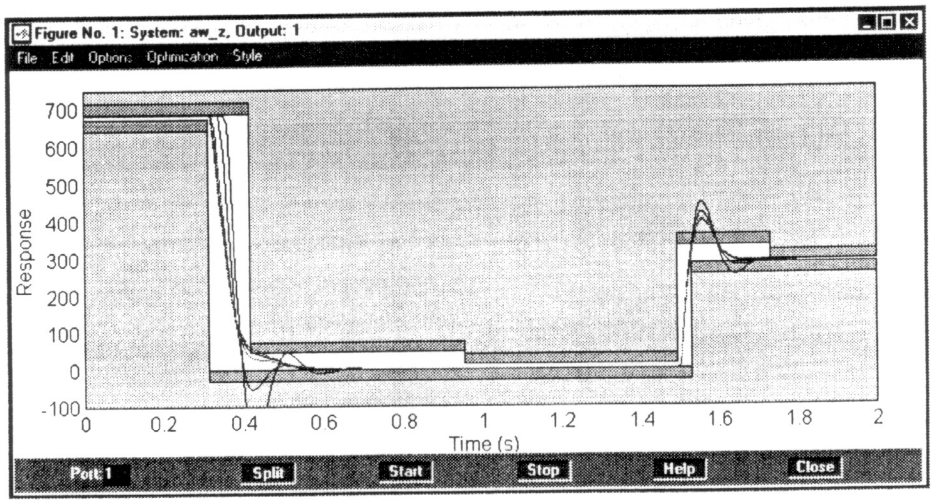


Fig. 10. NCD block for optimisation of PI controller parameters.

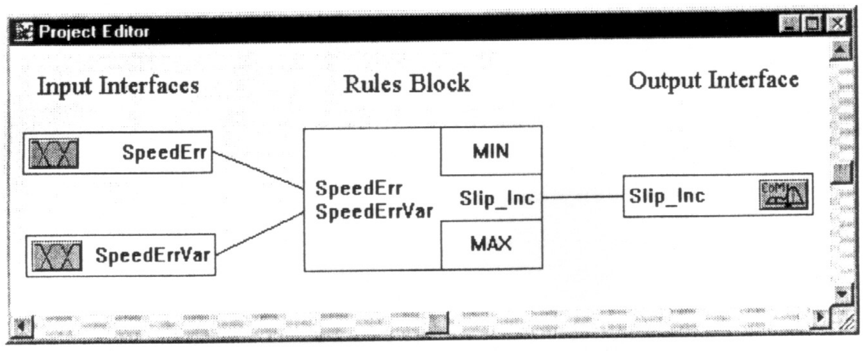


Fig. 11. Fuzzy logic controller structure.

project, linguistic variables, rules definition, and interactive debugging [14,15,17]. The use of the NeuroFuzzy Module to automatically generate the membership functions and rules was tried, but in this case, as stated in the introduction, the results were not very useful. Therefore, the selection of the rules and the tuning of the membership functions were mostly done by trial and error.

3.3.2.1. Project definition The first step when using the fuzzyTECH is to define the structure of the controller by means of the project editor window. Here one can define the inputs and outputs of the fuzzy logic system and how they should interact. The project editor displays the controller structure and allows the designer to directly access linguistic variables and rule definitions. Fig. 11 presents the fuzzy logic controller structure for the induction motor slip control system.

The small blocks on the left side are the input interfaces, which also contains the fuzzification of the input values. The large block in the middle of the screen is the rules block, which contains an independent set of fuzzy logic rules. The small block on the right side is the output interface, which contains the defuzzification method. The Project Editor also allows the access to all parts of the fuzzy logic system. A double click on the rule block, for example, opens an editor for the rules contained therein.

3.3.2.2. Linguistic variables definition The next step of the controller design is the definition of the linguistic variables. The graphic interface of the development tool allows the designer to easily create the most suitable linguistic variables and membership functions for the application. Fig. 12 shows the input and the output membership functions. The input membership functions are defined taking into account the motor speed error (SpeedErr) and the speed error variation (SpeedErrVar). The motor slip increment (Slip_Inc) is the output membership function.

Triangular Membership Functions (MFs) were employed for the inputs and Singleton Membership Functions (which can be considered as a special case of Triangular MFs) were employed for the output. *SpeedErr* uses 3 MFs: Negative (N), Zero (ZE) and Positive (P). *SpeedErrVar* is described with 5 MFs: Negative Large (NL), Negative Small (NS), Zero (ZE), Positive Small (PS) and Positive Large (PL). The output *Slip_Inc* uses 7 MFs: Negative Large (NL), Negative Medium (NM), Negative Small (NS), Zero (ZE), Positive Small (PS), Positive Medium (PM) and Positive Large (PL).

The method of defuzzification used was the CoM (Center of Maximum), which considers only the maximum value positions of the MFs. In this case the use of Singleton or Triangular MFs for the output produces the same results.

During the definition of linguistic variables, the development tool allows the user to define two representations for the variables: the ''shell value'' and the ''code value''. The shell values are the ''real-world'' values that the variables represent. They are only used to display actual data with the tool. The code values are the 16-bit internal values that the microcontroller uses to calculate results and range from 0 to 62535 (16bit representation).

3.3.2.3. Rules definition Fig. 13 shows the fuzzy controller rules. They were set according to the understanding of the behaviour of the system. One set of rules (rules 1, 2 and 10, 11) is included to provide rapid response to a large speed error (SpeedErr). In this case the slip increment (Slip_Inc) is large or medium (it is positive or negative depending on the error sign). Another set of rules (3, 4 and 8, 9) avoids motor speed overshoot: Slip_Inc is medium or small and its sign depends on the sign of SpeedErr and SpeedErrVar. Rules 5, 6 and 7 are included to maintain the speed error near zero (steady state rules): here the Slip_Inc is either small or zero and its sign depends on SpeedErr.

The fuzzy development tool with its spreadsheet rule editor offers an easy way to examine and define the suite of rules. It also permits the definition of rule aggregation (MIN and MAX operators are available) and rule composition (MAX and BSUM). For this application the operators used were MIN for aggregation and MAX for composition. The Degree of Support (DoS) for all rules, that is, the rule weight in the composition process, has been set to 1.

As stated before, the defuzzification method applied in this case was the CoM. In control applications CoM is most commonly used because the output value represents the best

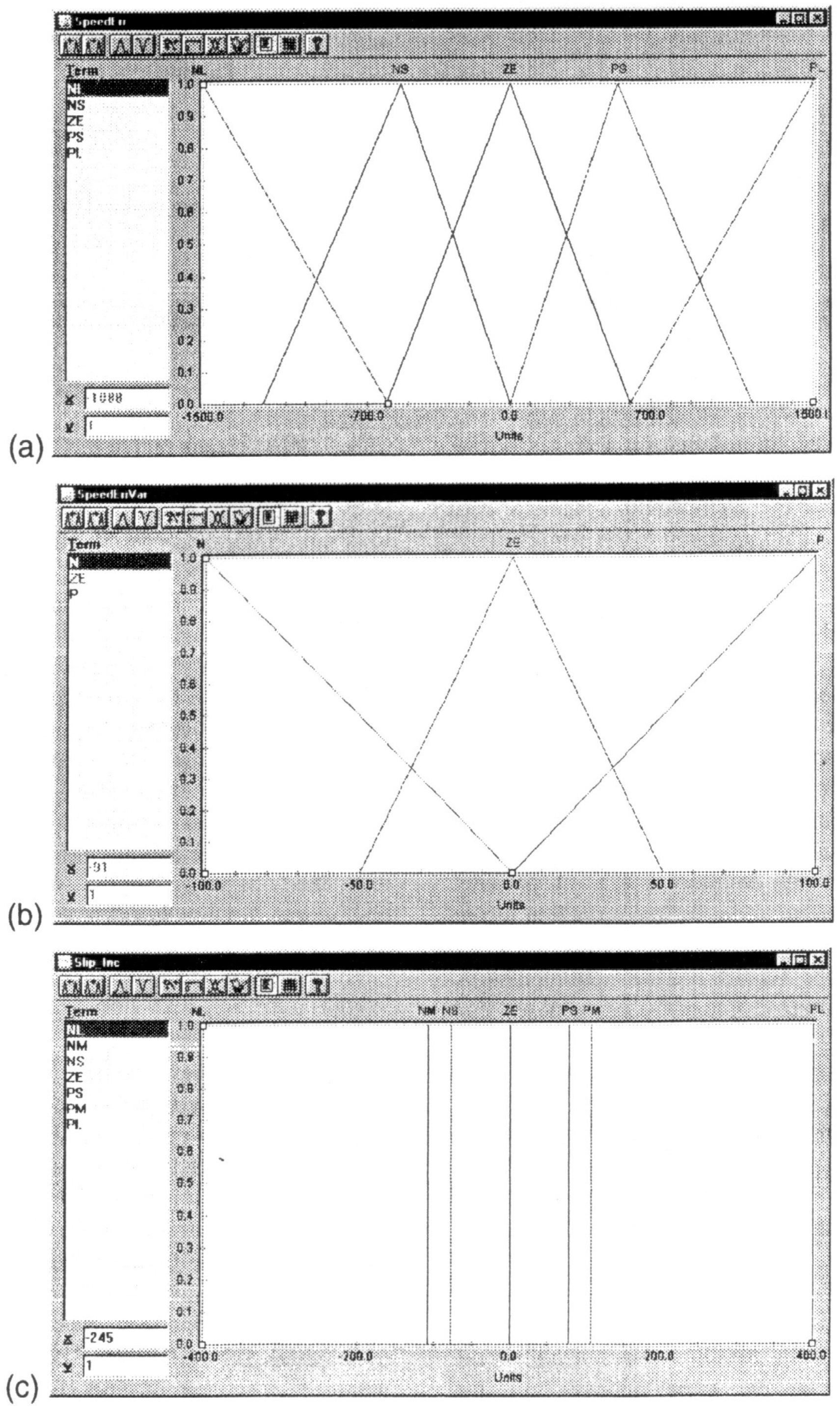
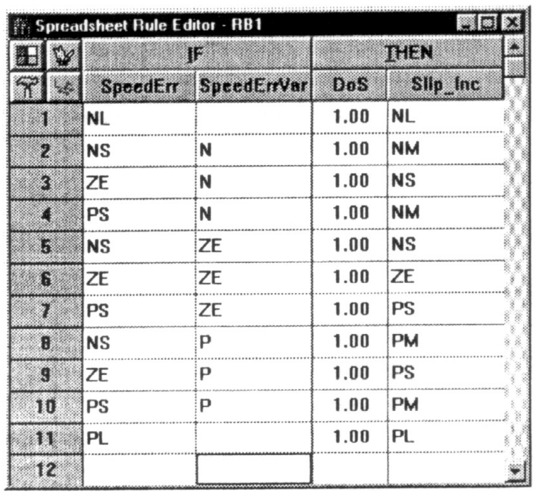


Fig. 12. Fuzzy logic controller Membership Functions: (a) *SpeedErr*; (b) *SpeedErrVar*; (c) *Slip_Inc*.

compromise of all inferred results, with high computational efficiency.

3.3.2.4. Interactive debugging This feature allows the user to visualise the entire fuzzy logic inference graphically. All the editors used for designing the system also display the computation results. It is possible to see how the input values are fuzzified, which rules fire to what degree, and how the fuzzification is computed. The most important issue is that most parts of the system can be modified while one can see how this affects the performance. If, for instance, the definition point of a membership function is moved, it is possible to see all the effects of this change on the rules and on the defuzzification.

To simulate the fuzzy logic control within the Matlab/Simulink environment, the PI controller block in Fig. 8



Spreadsheet Rule Editor - RB1

	IF		THEN	
	SpeedErr	SpeedErrVar	DoS	Slip_Inc
1	NL		1.00	NL
2	NS	N	1.00	NM
3	ZE	N	1.00	NS
4	PS	N	1.00	NM
5	NS	ZE	1.00	NS
6	ZE	ZE	1.00	ZE
7	PS	ZE	1.00	PS
8	NS	P	1.00	PM
9	ZE	P	1.00	PS
10	PS	P	1.00	PM
11	PL		1.00	PL
12				

Fig. 13. Spreadsheet rules editor.

was replaced by the "Fuzzy Controller Block", which is expanded in Fig. 14. Here, block A computes the error between the speed reference and the actual speed. Block B computes the error variation. The M-code produced by the fuzzyTECH MCU-96 tool is invoked by a Matlab function (M-File represented by block C), having the speed error and the speed error variation as inputs, and the slip increment as output. The shadowed blocks (D), implement an accumulator which produces the motor slip frequency. The remaining blocks are used for proper scaling and limitation of variables.

As said before (Section 2.3) the code for block C was automatically generated by the tool for the INTEL 80196 microcontroller used for the system implementation.

4. Simulation Results

The computational simulations compare the behaviour of PI and fuzzy logic controllers, showing speed and slip values during motor start-up and in response to a sudden load change from zero to nominal motor torque value.

First simulations were performed with the motor-load inertia value for which the PI controller parameters were optimised, which is the same value of the experimental implementation. Then the inertia value was modified in order to evaluate the controllers sensitivity to system parameters changes.

Fig. 15 presents the simulation results for the first inertia value ($J = 2 \times 10^{-2}$ kg m^2). The response for both controllers is almost identical, as can be better seen in Fig. 15b, where the speed scale is expanded. Fig. 16 shows simulation results with $J = 10 \times 10^{-2}$ kg m^2. In this case the fuzzy controller response is better: the overshoot values are smaller and its response is faster.

The controllers behaviour can be better compared using standard performance indexes. Table 1 shows the values of IAE (Integral of Absolute Error) and ITSE (Integral of Time Squared Error) for the PI and fuzzy controllers, during start-up and load application conditions. These indexes show that the fuzzy logic controller performs better than the PI controller when the motor-load inertia is changed to $J = 10 \times 10^{-2}$ kg m^2.

5. Experimental results

Experimental results were achieved with the induction motor coupled to an eddy current dynamometer. It was measured a total inertia value $J = 2 \times 10^{-2}$ kg m^2. In these experiments the motor is started-up unloaded and after about 3.0 s the nominal load torque is applied.

Fig. 17 and Fig. 18 show the speed and slip responses for the PI controller and for the fuzzy logic controller, respectively. These figures confirm that, for this inertia value, the response is almost identical for both controllers. When nominal load is applied the motor speed decreases by about 4.5% before the controllers manage to reestablish the speed to the reference value, increasing the slip.

The main difference between the PI and fuzzy controller responses has to do with their sensitivity to speed noise (which was not considered in the simulations): the fuzzy logic controller behaviour is clearly better, presenting a speed ripple of about 2.7% unloaded and 2.0% loaded, against values of respectively 5.6% and 3.0% for the PI controller.

Table 2 shows the standard performance indexes for the measured speed response of the two controllers. The fuzzy logic controller results are better.

6. Conclusions

An evaluation of fuzzy logic techniques applied to the control of induction motor was presented.

Both simulation and experimental results confirmed that the fuzzy logic approach is feasible and can be an interesting alternative to conventional control, even when the system model is known and linear.

The implemented fuzzy logic controller presented a slightly superior dynamic performance when compared with a more conventional scheme (PI controller with anti-windup mechanism), namely in terms of insensitivity to

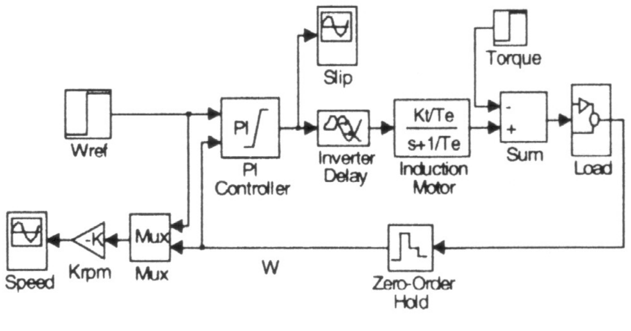


Fig. 8. Slip control system block diagram in the *Simulink* environment.

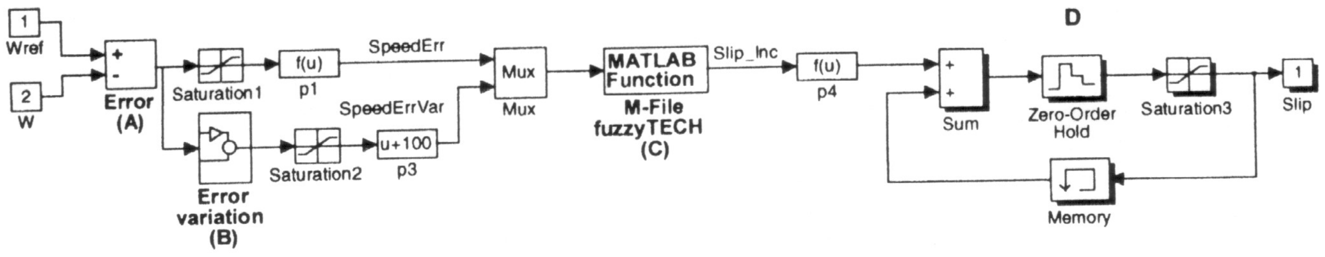


Fig. 14. Fuzzy controller block diagram expanded.

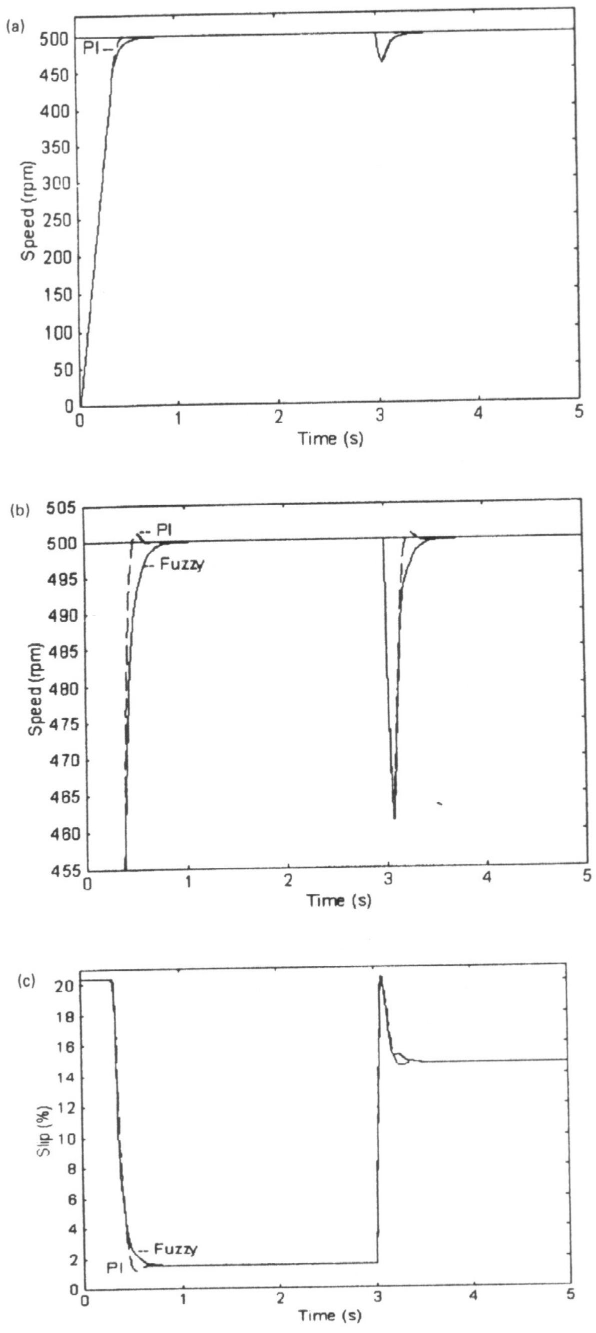


Fig. 15. PI and fuzzy controllers simulation response for $J = 2 \times 10^{-2}$ kg m^2: (a) Speed; (b) Expanded view of speed; (c) Slip.

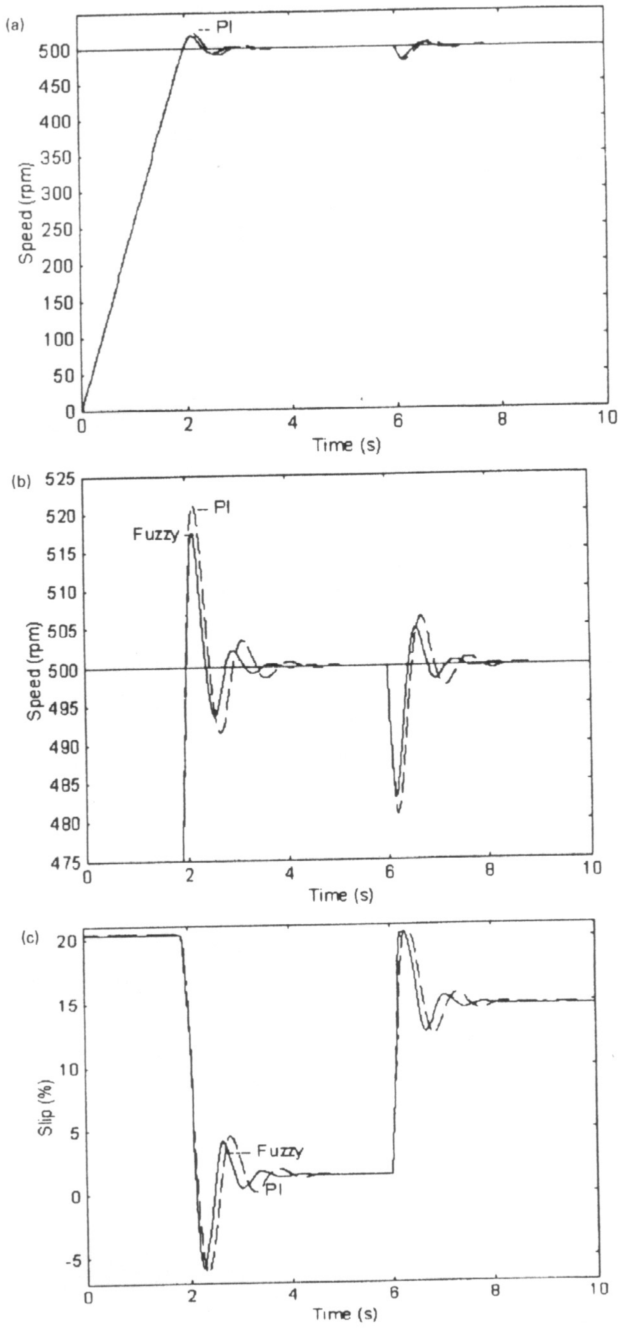


Fig. 16. PI and fuzzy controllers simulation response for $J = 10 \times 10^{-2}$ kg m^2: (a) Speed; (b) Expanded view of speed; (c) Slip.

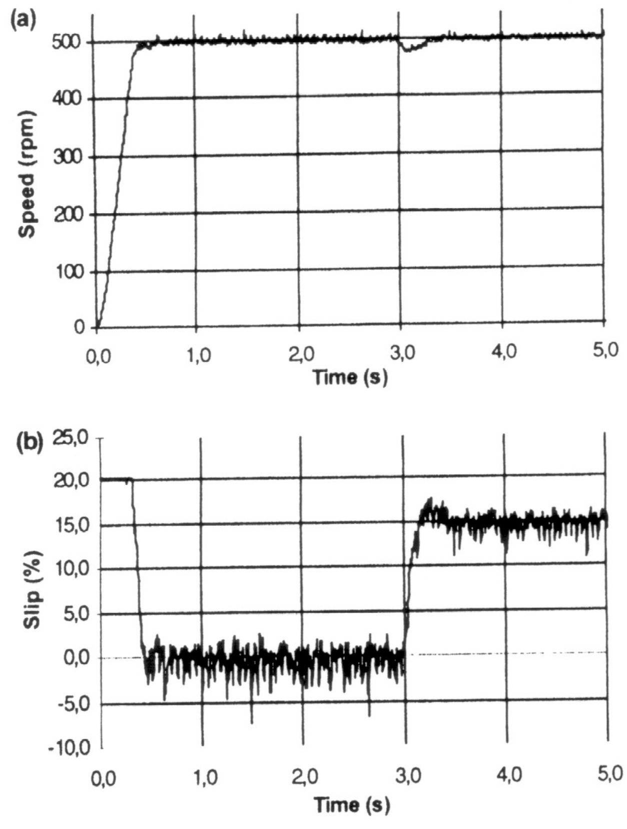


Fig. 17. PI controller experimental results: (a) Speed; (b) Slip.

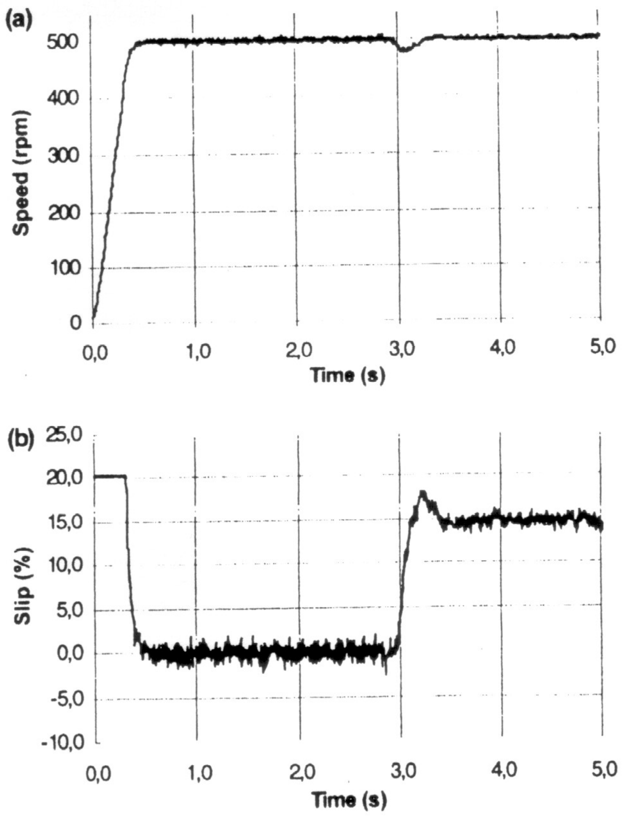


Fig. 18. Fuzzy logic controller experimental results: (a) Speed; (b) Slip.

changes in model parameters and to speed noise. This can be an important requirement in speed/position control schemes using electrical machines, namely in robotics.

Some authors claim that fuzzy logic controllers are easier to tune than conventional ones, and therefore the development times are shortened. From the experience of the authors this statement cannot be supported, at least for this type of application and with the tuning process done manually. Progress in research on machine learning techniques may change this negative point. A lot should be expected from the development of these techniques.

Matlab/Simulink and fuzzyTECH proved to be two compatible and powerful software tools, respectively for simulation and controller design.

The hardware used to accomplish the system was minimum. No fuzzy processors or any other specific hardware was used. A standard Intel 80C196KD microcontroller performs the control algorithms and generates the PWM waveforms for the IGBT motor drive inverter.

Table 1
Simulation speed response performance $-J = 10 \times 10^{-2}$ kg m^2

	Start-up		Load application	
	IAE	ITSE	IAE	ITSE
PI	1.0240	2.2532	0.0180	0.0162
Fuzzy	1.0182	2.2565	0.0122	0.0100

Table 2
Measured spped response performance $-J = 2 \times 10^{-2}$ kg m^2

	STart-up		Load application	
	IAE	ITSE	IAE	ITSE
PI	0.2506	0.0182	0.0155	0.0014
Fuzzy	0.2191	0.0143	0.0137	0.0012

References

[1] J.M. Mendel, Fuzzy Logic systems for engineering: A tutorial, Proceedings of the IEEE, 83 3, 1995, pp. 345–377.
[2] R Jager, Fuzzy logic in control, PhD Thesis, Delft university, Holland, 1995.
[3] G.C. Sousa, B.K. Bose, J.G. Cleland, Fuzzy logic based on-line efficiency optimisation control of an indirect vector controlled induction motor drive, IEEE, Trans. On industrial electronics 42 (2) (1995) 192–198.
[4] J. Fonseca, J.L. Afonso, J.S. Matins, C. Couto, Evaluation of neural networks and fuzzy logic techniques applied to the control of electrical machines, Proceedings of the 5th UK mechatronics forum international conference,Portugal, 2, 1996, pp. 15–20.
[5] J.L. Afonso, J. Fonseca, J.S. Martins, C. Couto, Fuzzy logic techniques applied to the control of a three-phase induction motor, ISIE'97-IEEE International symposium on Industrial Electronics, Guimarães,

Portugal, July 7-11, 1997, IEEE Catalog number: 97TH8280, ISBN: 07803-3936-3, pp. 1179–1184.
[6] J.H. Holland, Adaption in natural and artificial systems, Univ. of Michigan press, Ann Arbor, MI, 1975.
[7] C. Couto, J.S. Martins, Control of a voltage source inverter fed induction motor with on-line efficiency optimisation, IEEE ICIT'94, Guangzhou, China, 1994, pp. 528–532.
[8] J.S. Martins, Controlo de Velocidade do motor de InduÇão Trifásico, PhD Thesis, University of Minho, Portugal, 1993.
[9] W. Leonard, Control of electrical drives, Springer-Verlag, Berlin Heidelberg, New York, 1985.
[10] A. Cavallo, R. Setola, F. Vasca, Using Matlab, Simulink and control system toolbox: A practical approach, Prentice Hall, Europe, UK, 1996.
[11] MATLAB: High-performance Numeric Computation and Visualization Software-Reference Guide, The MathWorks Inc., April 1993.
[12] SIMULINK:The dynamic system simulation software-user's guide, MathWorks Inc., April 1993.
[13] A.F. Potvin, Nonlinear control design toolbox, The Math Works Inc., 1993.
[14] fuzzyTECH reference manual, Inform Software Corporation, GmbH, 1996.
[15] C.V. Altrock, Fuzzy Logic and Neuro Fuzzy applications explained, Prentice-Hall, UK, 1995, pp. 63–81.
[16] K.J. Astrom, Computer controlled systems, Prentice-Hall, UK, 1990.
[17] P. Guillermin, Fuzzy logic applied to motor control, IEEE Trans. on industry applications 32 (1) (1996) 51–56.