Chapter 5

Modelling and Implementation of a Series DC Motor Drive System

Angelo José Junqueira Rezek, Carlos Alberto Murari Pinheiro, Tony Youssif Teixeira Darido, Valberto Ferreira da Silva, Otávio Henrique Salvi Vicentini, Wanderson de Oliveira Assis and Rafael Di Lorenzo Corrêa

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/59814

1. Introduction

For the purpose, of controlling a DC motor in a mono quadrant, a totally controlled Graetz converter bridge (a static converter) was used. For the implemented system, a microcomputer, a data acquisition board, an electronic firing circuit, as well as a current and speed sensors were used.

In this computer study, C++ language was used for the implementation of a controlled DC drive using a DC series motor.

Through the computer, the system control was performed directly via software and a data acquisition board containing A/D and D/A converters.

The board used was a PCL-711B PC-Multilab card from Advantech Co. (www.advantech.com), with A/D and D/A conversion, as well as digital inputs and outputs. The A/D conversion had 12 bits resolution, with eight input channels programmable for input ranges of ±5 [V] and a conversion time of 25 [μs]. The D/A conversion had the same resolution (12 bits) but with a single output channel that had an accommodation time of 30 [μs] and output ranges from 0 to +5 [V].

Two input channels were used for A/D conversion, corresponding to the speed and current feedback signals; one output channel D/A was used for the control signal (V_{CC}).

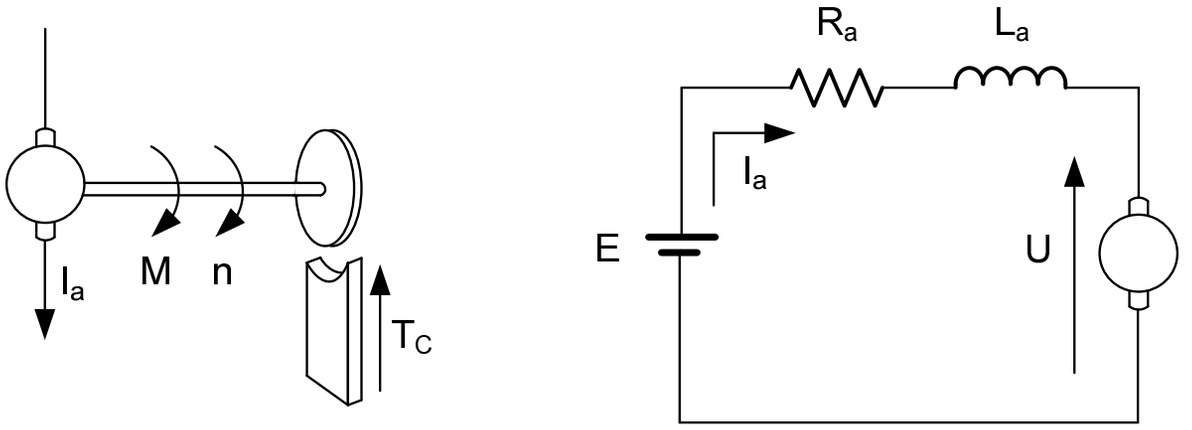
The control software was responsible for the acquisition of input data, for the A/D and D/A conversions, implementation of the control algorithm for the speed and current grids, and for

generation of the control signal for the firing circuit. The program also included an on-line parameters setting. An output DC signal of the card (0-10 [V]) was used as the input of the firing circuit in order to control the series DC motor and for this purpose, two regulators, connected in a cascading fashion in terms of speed and current loops, was projected. The load torque was imposed using a three phase synchronous generator, connected to the same shaft of the series DC motor and supplying a three-phase resistor bank.

2. Motor block diagram

2.1. Mechanical part equating

Figure 1 shows the series DC motor armature circuit. The diagram of the drive system's mechanical part is also presented.



U: Counter electromotive force [V]
E: Applied voltage [V]
R_a, L_a: Total resistance and inductance of the armature circuit
Φ: Motor flux
Ia: Armature current
M: Motor torque
T_c: Load torque
J: Moment of inertia (motor + load)
n: Speed [rpm]
ω: Rotation [rad / s]

Figure 1. Motor armature circuit and diagram for the drive system mechanical part, being:

Also:

$$M = K_1 \Phi I_a$$

Being I_a the armature current average value and also, $K_1\Phi = KI_a$, therefore:

http://dx.doi.org/10.5772/59814

$$M = KI_a^2 \tag{1}$$

For the accelerating torque:

$$M - T_c = B \tag{2}$$

$$B = J\frac{dw}{dt} \tag{3}$$

$$\omega = \frac{2\pi}{60} n_0 \frac{n}{n_0} \tag{4}$$

Being:

n_0: No load speed [rpm]

M_n : Rated torque.

$$B = M_n \frac{B}{M_n} \tag{5}$$

(4) e (5) into (3) results:

$$\begin{aligned} M_n \frac{B}{M_n} &= \mathrm{J}\ \mathrm{d}\frac{\frac{2\pi}{60} n_o \frac{n}{n_o}}{dt} \\ M_n \frac{B}{M_n} &= \mathrm{J}\ \frac{2\pi}{60}\frac{n_o}{n_o}\frac{dn}{dt} \\ n &= \frac{n_0}{M_n}\frac{1}{\frac{2\pi}{60} J \frac{n_0}{M_n}}\int B.dt \end{aligned} \tag{6}$$

By defining the acceleration time constant $T_{H,}$ as:

$$T_H = \frac{2\pi}{60}\frac{Jn_0}{M_n} \tag{7}$$

This time constant may be interpreted as the time required for the motor to reach no load speed from a resting state, when it is accelerated by a resultant torque equal to the rated motor torque.

$$n = \frac{n_0}{M_n}\frac{1}{T_H}\int B.dt \tag{8}$$

Figure 2 illustrates the block diagram related to the mechanical motor part.

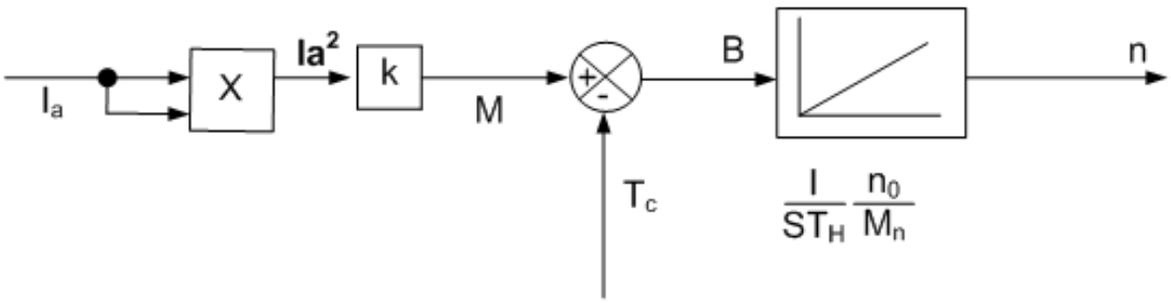


Figure 2. Block diagram of the drive mechanical part.

Setting the magnitude values in pu, current, load torque, accelerating torque, speed and motor torque results in:

$$\frac{I_a}{I_N} = i_a \text{ (pu)}$$
$$\frac{T_c}{M_n} = t_c \text{ (pu)}$$
$$\frac{B}{M_n} = b \text{ (pu)}$$
$$\frac{n}{n_o} = n_u \text{ (pu)}$$
$$\frac{M}{M_n} = m \text{ (pu)}$$

The block diagram in Figure 2 can be represented in pu as shown in Figure 3.

2.2. Armature circuit equating

$$E = R_a I_a + L_a \frac{dI_a}{dt} + U$$

http://dx.doi.org/10.5772/59814

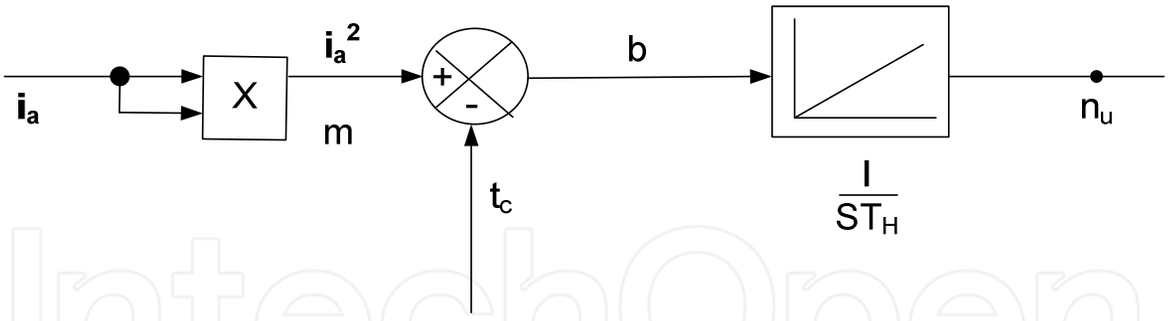


Figure 3. Representation of the motor mechanical part in pu.

Applying the Laplace transform results in:

$$E(S) = R_a I_a(S) + SL_a I_a(S) + U(S)$$
$$E(S) - U(S) = I_a(S)\left[R_a + SL_a\right]$$

$$I_{a(S)} = \frac{E(S) - U(S)}{R_a + SL_a} \tag{9}$$

Defining:

$$T_a = \frac{L_a}{R_a} \tag{10}$$

$$I_a(S) = \frac{E(S) - U(S)}{1 + ST_a} \times \frac{1}{R_a} \tag{11}$$

Regrouping:

$$\frac{I_a}{I_N} \times I_N = \text{E-U}\,\frac{E_N}{R_a}\,\frac{E_N}{E_N} \times \frac{1}{1 + ST_a}$$
$$\frac{I_a}{I_N} = \frac{\text{E-U}}{E_N R_a}\,\frac{E_N}{I_N} \times \frac{1}{1 + ST_a}$$
$$\frac{I_a}{I_N} = \frac{\text{E-U}}{E_N}\,\frac{E_N}{R_a I_N} \times \frac{1}{1 + ST_a}$$

The armature current I_a appears to be normalized by the nominal current I_N. Voltages E and U are also normalized by the nominal voltage E_N, setting the normalized values as:

$$i_a = \frac{I_a}{I_N} \tag{12}$$

$$e = \frac{E}{E_N} \tag{13}$$

$$u = \frac{U}{E_N} \tag{14}$$

$$V_i = \frac{E_N}{R_a I_N} \tag{15}$$

The block diagram of the armature circuit l is shown in Figure 4.

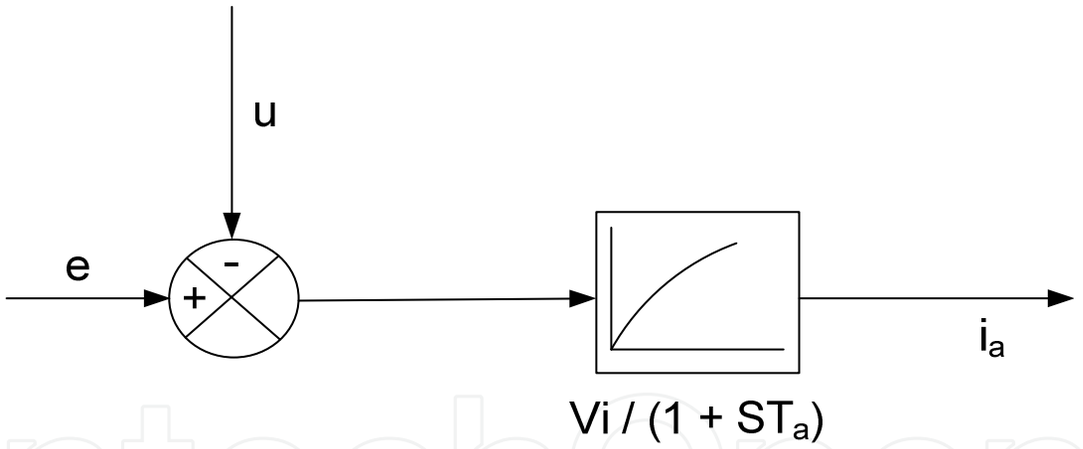


Figure 4. Block diagram of the DC motor armature circuit.

The factor $\frac{V_i}{1+ST_a}$ can be considered a delay element of 1 ª order. Time constant Ta can be determined in two ways:

a. Measuring La and Ra;

b. Applying a reduced voltage step in the armature circuit when the rotor of the motor is blocked.

Figure 5 shows the armature circuit.

http://dx.doi.org/10.5772/59814

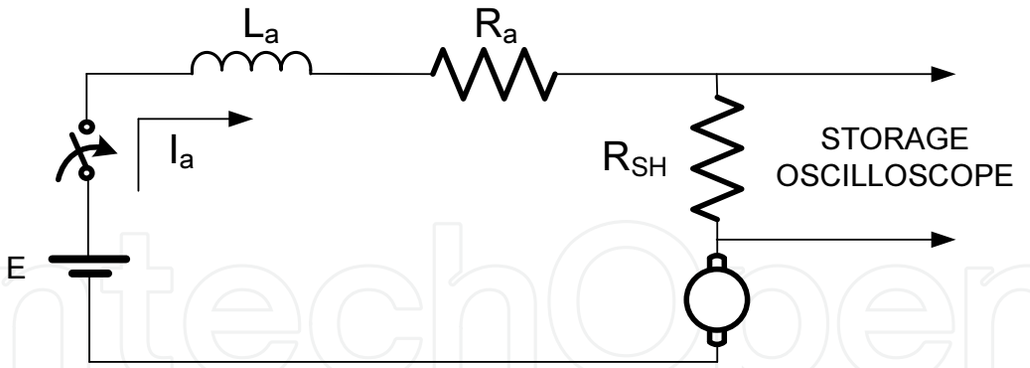


Figure 5. Reduced voltage step applied to the armature circuit.

The current I_a is captured through an oscillograph or storage oscilloscope (voltage drop in the shunt resistor R_{SH}). Figure 6 illustrates the expected time transient response. A current clamp can also be used, instead of the resistor $R_{SH.}$

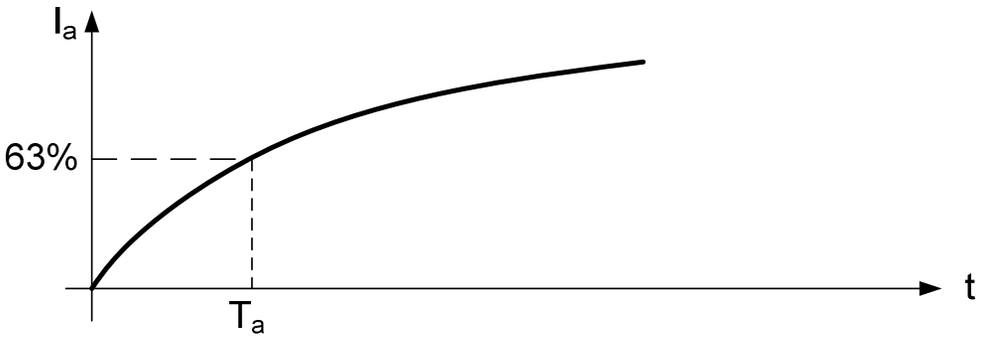


Figure 6. Current response for a voltage step applied to the armature circuit.

Marking up to 63% of the regime value and checking the corresponding time on the horizontal axis, yields the time constant τ_a, as shown in Figure 4.

$$i_a = \frac{e-u}{1++ST_a} V_i \tag{16}$$

Considering an interpretation of the constant V_i (equation 15) results in a motor with the rotor locked and with a rated voltage applied to the armature.

$$I_{AK} = \frac{E_N}{R_a} \tag{17}$$

$$\frac{I_{AK}}{I_N} = \frac{E_N}{R_a I_N} \tag{18}$$

Comparing (18) e (15), results in :

$$I_{AK} = V_i I_N \tag{19}$$

The V_i factor can be interpreted as the multiplying factor of the rated current for gaining a current with the rotor locked when nominal voltage is applied to the armature circuit (starting current in pu).

Figure 7 shows a schematic diagram of the series DC motor with rated flow and considering the normalized magnitudes (pu).

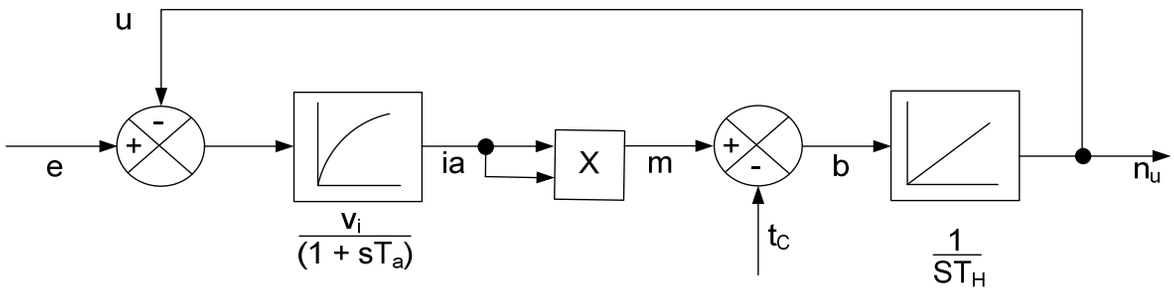


Figure 7. Schematic block diagram of the series DC motor.

3. Complete block diagram including regulators, filters and transducers

Figure 8 shows the controlled drive complete block diagram.

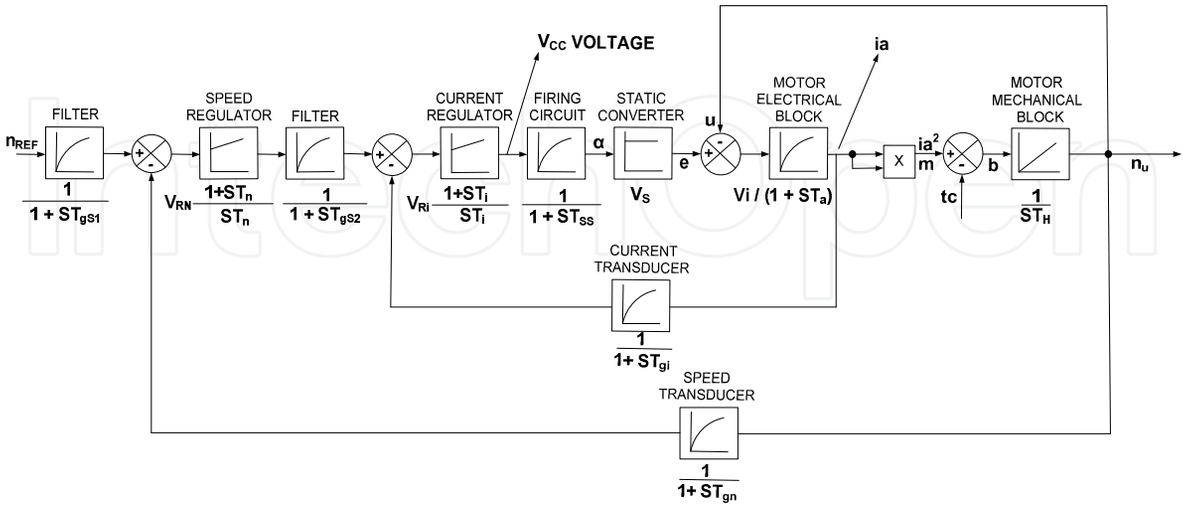


Figure 8. Controlled drive complete block diagram.

Where:

http://dx.doi.org/10.5772/59814

τ_{gs1}: Filter time constant of the reference channel of the speed loop

τ_{gs2}: Filter time constant of the reference channel of the current loop

V_{RN}: Gain of the speed regulator

V_{Ri}: Gain of the current regulator

τ_n: Time constant of the speed regulator

τ_i: Time constant of the current regulator

τ_{gn}: Filter time constant of the speed transducer

τ_{gi}: Filter time constant of the current transducer

τ_{ss}: Time constant of the firing circuit

V_s: Gain of the static converter

The regulators in Figure 8 were considered as PI (proportional integral) type and its parameters will be determined by the design procedure.

4. Design of the current regulator and filters of the current control loop

Figure 9 illustrates the current regulation loop.

Series DC motor data:

- - Power: 1.7 [KW]
- - Current: 7.72 [A]
- - Speed: 1500 [rpm] (Rated)
- - No load speed: 1770 [rpm]
- - Voltage: 220 [V]
- - ΣRa = 7.0 [Ω] (Total resistance of the armature circuit)
- - ΣLa = 490 [mH] (Total inductance of the armature circuit, including the smoothing reactor)
- - $\tau_H = 1.2[s]$ - (Accelerating time constant)

The current transducer is a diode bridge that supplies resistors connected to the AC (alternating current) side and feeds through the secondary current transformers (CTs), 30/5 [A]. Figure 10 shows the current transducer.

A current hall sensor can also be used as current transducer in DC side.

Where: L_d: Smoothing reactor inductance

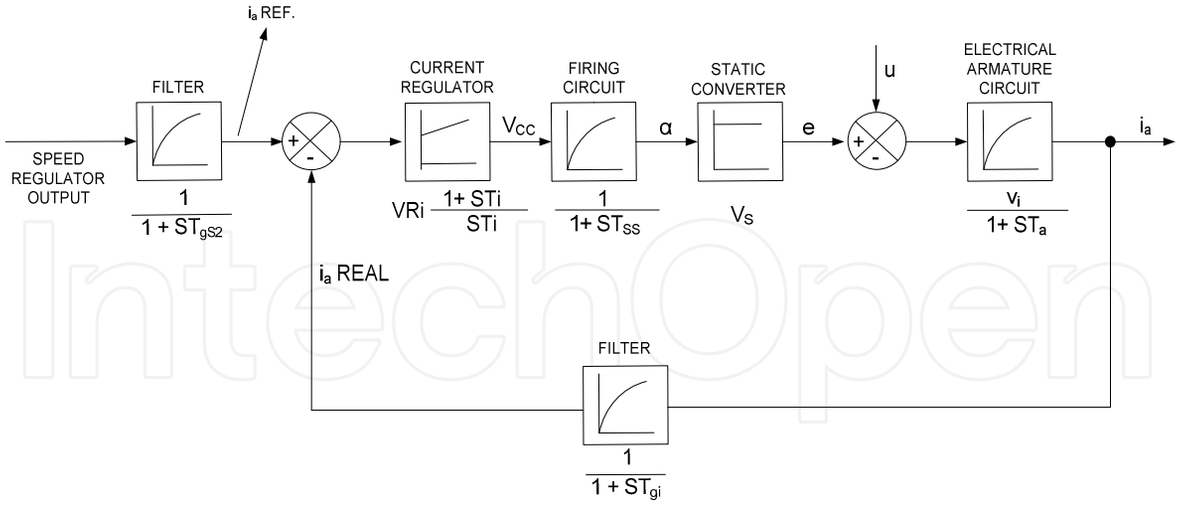


Figure 9. Current regulation loop.

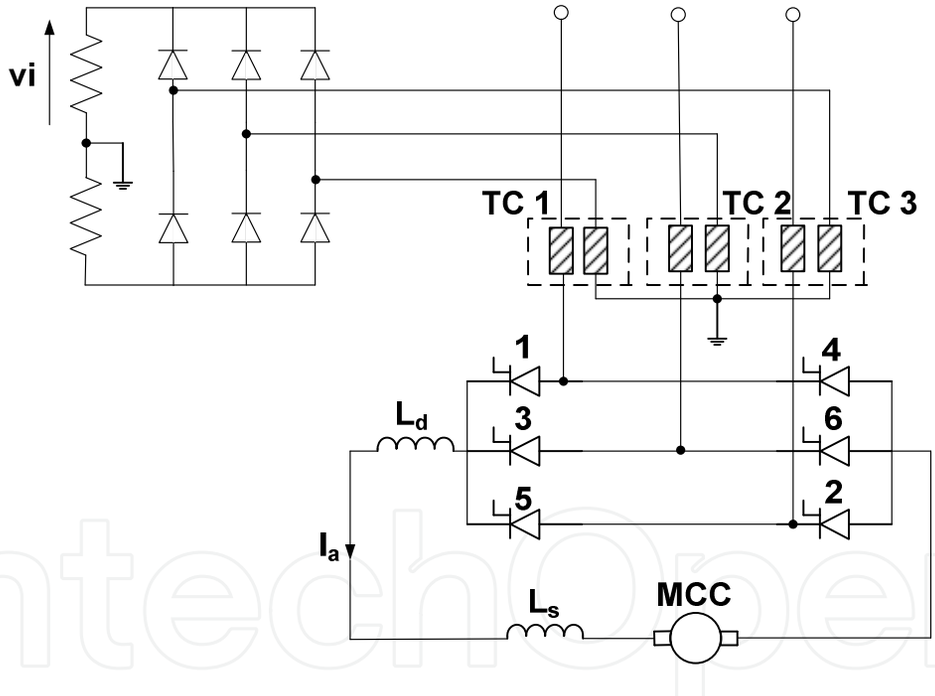


Figure 10. Current transducer.

L_s: Series field inductance

The V_i signal (current transducer) has, as is known, a (1/6) cycle ripple wave. It should have:

$$\tau_{gi} \leq \frac{1}{2} \frac{Period}{N^o ofpulses}$$

http://dx.doi.org/10.5772/59814

For 60 [Hz], a period of 16.7 [ms] and six pulse bridges, the following is adopted:

$$\tau_{gi} = 1.5 \left[\text{ms} \right]$$

The firing circuit cannot instantly respond to the change in the firing angle α. This time constant can vary by a range of zero to one sixth of a cycle. The value τ_{ss} = 2.5 [ms] will be used. The time constant τa was obtained measuring La and Ra.

$$\tau_a = \frac{L_a}{R_a} = 70 \ [ms]$$

This constant has been calculated taking into account the total inductance La in the series with the armature circuit, which corresponds to the inductances of machine (armature, interpoles and series field) added to the inductance of the external smoothing reactor.

The Vi constant is equal to:

$$V_i = \frac{E_n}{R_a I_a} = 4.07$$

The Vs gain of the converter is obtained, as follows:

$$E = 1.35 U_2 sin\alpha$$

Being U_2, the AC supply voltage of converter bridge is:

$$\frac{dE}{d\alpha} = -1.35 U_2 sin\alpha$$

Multiplying member by member by $\frac{\pi}{E_N}$, results:

$$\frac{d\left(\frac{E}{E_N}\right)}{d\left(\alpha / \pi\right)} = -1.35 \frac{U_2}{E_N} \pi sin\alpha$$

The value of U_2 is determined for the DC side-rated voltage as being equal to (220V), considering the value of the firing angle at 30°. This procedure results in U2 = 188 [V].

Also:

$$\frac{de}{d\alpha_2} = -1.35 \times \frac{188}{220} \times \pi sin\alpha$$
$$\frac{de}{d\alpha_u} = -1.15\pi sin\alpha \tag{20}$$

Being As the value in (pu) of the firing angle α, is equal to $\alpha_u = \frac{\alpha}{\pi}$

Results:

$$Vs = \left|\frac{de}{d\alpha_2}\right| = 1.15\pi sin\alpha$$

When α = 90 ° the maximum gain is:

$$p/_{\alpha=90^o} = 1.15\pi$$
$$Vs_1 = \left|\frac{de}{d\alpha_2}\right|$$

For α = 30°, gain is given by:

$$p/_{\alpha=30^o} = 1.15 \times \pi \times 0.5$$
$$Vs_2 = \left|\frac{de}{d\alpha_2}\right|$$

The average gain, $V_{s,}$ can be determined as:

$$Vs = 2.71 \tag{21}$$

The current regulation system gain is then obtained as:

$$Vsia = Vs \times Vi = 2.71 \times 4.07 = 11.03 \tag{22}$$

The sum of the small time constants is:

$$\sigma = \tau_{ss} + \tau_{gi} = 4\ [ms]$$

http://dx.doi.org/10.5772/59814

The relationship $\frac{\tau_a}{4\sigma}$ =4.37>1

According to Table 6.3 (pp 339) in *Introduction to Electronic Control* (Spanish) [4] by Friedrich Fröhr and Fritz Orttenburguer, the regulator type PI should be used (designed by type *cake recipe*).

According to Table 6.4 (pp 341) of the same text, the gain and the time constant of the regulator can be obtained thus:

$$V_{Ri} = \frac{\tau_a}{2V_{sia}\sigma} = 0.8 \tag{23}$$

$$\tau_i = 4\sigma \frac{\tau_a}{\tau_a + 3\sigma} = 13.66\left[ms\right] \tag{24}$$

The time constant value of the reference channel filter in [ms] is:

$$\tau_{gs2} = 4\sigma\left(1 - e^{-\left(\frac{\tau_a}{4\sigma - 1}\right)}\right) = 15.84\left[ms\right] \tag{25}$$

Thus, in summary:

CURRENT REGULATOR

- - Type: PI
- - Gain: VRi = 0.8
- - Time constant τi = 13.66 [ms]
- - Time constant value of the reference channel filter, τgs2 = 15.84 [ms]
- - Time constant value of the feedback channel filter, τ_{gi} = 1.5 [ms]

5. Design of the regulator and filters of the speed control loop

5.1. Design of the speed regulator

The current loop will be replaced by a corresponding retarder block of the first order, whose time constant, according to [4] is obtained thus:

$$\tau_e = 2\sigma + \frac{1}{2}\tau_{gs2} = 15\left[ms\right] \tag{26}$$

This results in a block diagram for the speed control loop as shown in Figure 11.

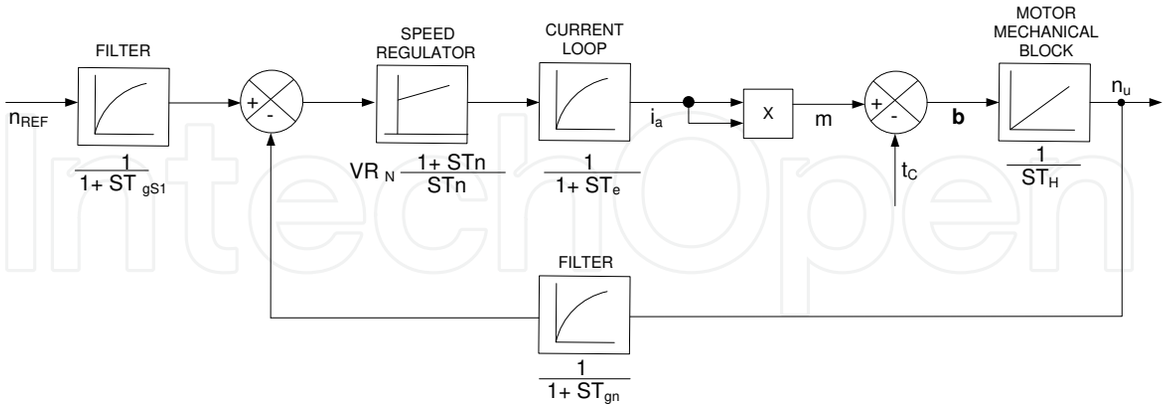


Figure 11. Speed loop regulation.

A speed transducer filter will be needed, for which the following time constant is adopted:

$$\tau_{gn} = 100\ [ms]$$

In the

The speed regulation loop contains the following time constants:

$$\tau_H = 1.2\left[s\right]$$
$$\tau_e = 15\left[ms\right]$$
$$\tau_{gn} = 100\left[ms\right]$$

The small time constants τe e τgn, will be grouped by an equivalent time constant σ′:

$$\sigma^{'} = \tau_e + \tau_{gn} = 115\left[ms\right]$$

The relationship, $\frac{\tau_H}{4\sigma^{'}}$=4.56>1 and in accordance with Table 6.3 [4], a PI type regulator should be used (designed by type *cake recipe*).

According to Table 6.4 [4] the gain and the time constant of the regulator are:

$$V_{RN} = \frac{\tau_H}{2\sigma^{'}} = 5.2 \tag{27}$$

http://dx.doi.org/10.5772/59814

The speed regulator time constant, is:

$$\tau_n = 4\sigma' = 460\left[ms\right] \tag{28}$$

5.2. Design of the speed reference and transducer feedback channels filters

The speed transducer is a DC tachogenerator coupled to the shaft of the DC motor. A filter is used due to the ripple wave of the tachogenerator output voltage. The adopted time constant is:

$$\tau_{gn} = 100\left[ms\right] \tag{29}$$

The speed reference channel filter has the following time constant value [4]:

$$\tau_{gs1} = 4\sigma' = 460\left[ms\right]$$

Thus, in summary, one has:

SPEED REGULATOR

- - Type: PI
- - Gain: VRN = 5.2
- - τn = 460 [ms]
- - Time constant value of the reference channel filter, τgs1 = 460 [ms]
- - Time constant value of the feedback channel filter, τ_{gn} = 100 [ms]

6. Ramp type firing circuit

The firing system used is a ramp type, implemented with the TCA 780 integrated circuit and manufactured by Siemens, as shown in Figure 12.

The intersection of the DC level with a ramp, which is internally generated in the TCA 780 integrated circuit, produces the pulses. Voltage VCC is the output of the current regulator, as shown in Figure 8. Three TCA 780 integrated circuits were used to produce six firing pulses for the thyristorized GRAETZ converter bridge, which are P1 and P2 for thyristors 1 and 4, P3 and P6, for thyristors 3 and 6 and P5 and P2 for thyristors 5 and 2, respectively. This process explains the firing circuit pulse generation stage. The other pulses include: enlargement of pulses, galvanic isolation of pulses and pulse amplification, the circuit description and explanation for which were not the objective of this work.

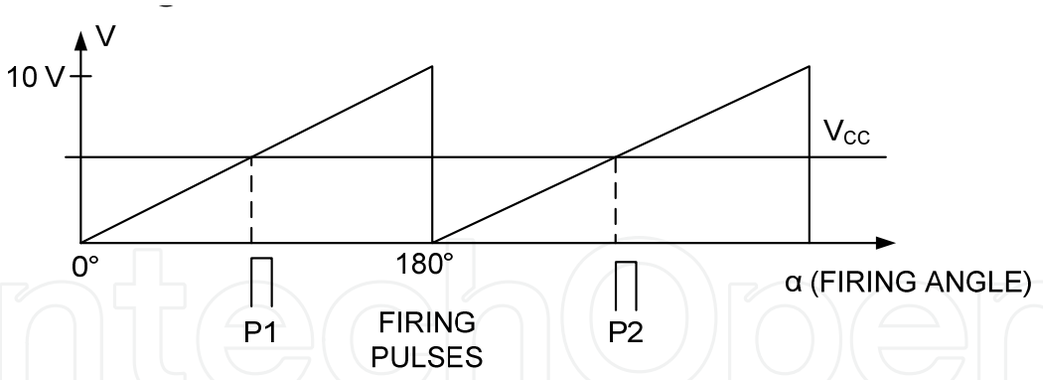


Figure 12. Ramp type firing circuit.

7. Control using fuzzy logic

This procedure is based on research provided in [10].

7.1. Fuzzy regulator

With the objective of understanding how this type of regulator works, it is helpful to understand the basics of fuzzy logic. Based on the theory of the fuzzy sets proposed by ZADEH in 1965 [9], this type of logic has proven to be one of the most interesting technologies for application in sophisticated control systems, providing a simple approach to decreasing costs and increasing the efficiency of such systems.

This chapter provides an introduction to this technology in the controlled drive of a DC machine with series excitation aimed at speed control and current limitation. According to [8], this type of system (a non-linear process) is better controlled with fuzzy controllers as opposed to conventional compensators.

7.2. Fuzzy control

The theory for control using fuzzy logic characterizes the variables of interest through linguistic expressions such as "very large", "large", "small", "hot", "cold", etc. These linguistic expressions are numerically represented by fuzzy sets, each set being characterized by a pertinence function varying from 0 to 1. The fuzzy control algorithm performs control actions written in terms of these imprecise ideas. The input variables of the fuzzy controller considered of interest (current regulator) are obtained by:

$$IE(k) = I_{real}(k) - I_{ref}(k) \tag{30}$$

$$PE(k) = GEi \times IE(k) \tag{31}$$

http://dx.doi.org/10.5772/59814

$$IIE(k) = IIE(k-1) + GVi \times IE \times T \tag{32}$$

Where IE(k) = Error of current; IIE(k) = Current error integral

PE(k) = Proportional error

GEi = Proportional gain; GVi = Integral gain (inverse of the regulator time constant)

$I_{real}(k)$ = Feedback current

$I_{ref}(k)$ = Reference current

u(k) (see program listing) = Vcc = Output control;

k = Sampling interval

T = Sample time

The pertinence functions are shown in Figure 13; variables are expressed in "pu",. The universe is adopted according to operational conditions.

Each control and action variable is decomposed into a set of fuzzy sets called labels. Generally, each label or fuzzy subset has an asymmetric form, with a trend toward a larger concentration close to the origin of cartesian axis. This allows for greater precision in the control close to the steady operation point.

The number of labels associated with one given variable must be an odd number between 5 and 9. Additionally, the end of each label must overlap those of neighbouring labels. This overlapping provides the fuzzy controller with continuous and steady action. The overlapping must be between 10 and 50% of the neighbouring area and the sum of the vertical points of the overlapping must preferably be less than 1.

The control does not require an accuracy modelling. The model can be unknown or badly defined.

The components of a conventional and of a fuzzy control system are more or less similar. The differences reside in the fact that in the fuzzy system, there is an element that converts the inputs into their fuzzy representations, i.e., the fuzzifier and another that converts the fuzzy outputs inferred into a solution, a numerical and precise value, i.e., the defuzzifier. In a fuzzy system, the value of an input is converted by the fuzzifier. Afterwards, control rules that are met are performed. This process produces a new fuzzy set representing each output or variable solution. The defuzzifier creates a value for the variable output of this new fuzzy set. The output value actuates the physical system. The change is picked up by a sensor and the control is restarted.

The fuzzy variables are defined at a numerical interval commonly called the "universe of discourse". The fuzzy rules are typically of the conditional form IF-THEN, as follows:

$$IF\left("x" \text{ is A and } "y" \text{ is B}\right) \text{ THEN}\left("z" \text{ is C}\right) \tag{33}$$

Where “x” and “y” are fuzzy variables and A, B and C are fuzzy subsets in the universe of discourse X, Y and Z, respectively. If the condition expressed in the rule is met, then the action specified is performed. To design a fuzzy controller, a series of rules must be built.

In this case, the error (IE), the error integral (IIE) and the control signal (V_{CC}) are considered fuzzy variables, with possible values given by pertinence functions (μ) to fuzzy sets such as small positive, small negative, zero and so on.

The pertinence functions for the error variable can be represented according to Figure 13.

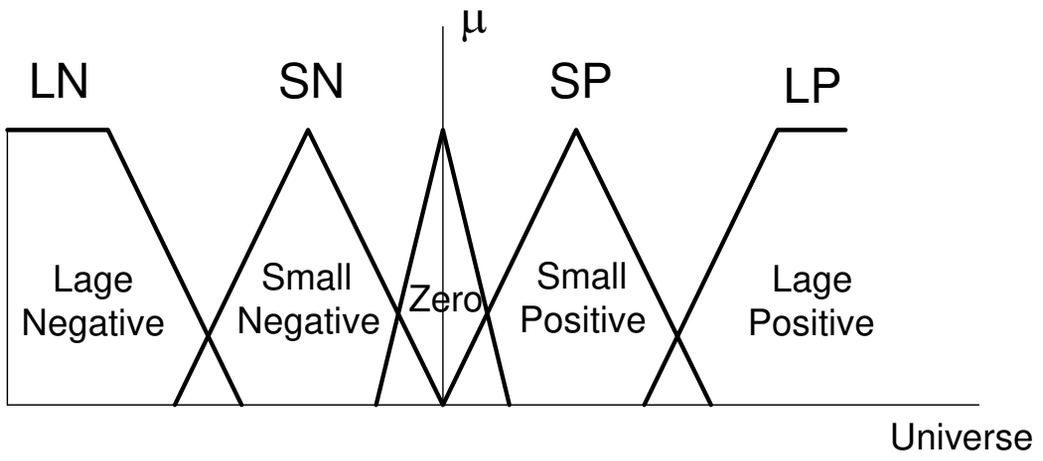


Figure 13. Representation of the typical pertinence functions for a fuzzy variable.

7.3. Design of the fuzzy controller — Current regulator

Figure 14 shows a representation of the pertinence functions.

The elaboration of rules can be based on the intuition and experience of the designer. The number of rules is related to the number of control variables. For the case under study, there were two control variables, each divided into five fuzzy subsets, producing 25 possible input combinations. Thus, 25 rules may be necessary. Here, some care must be taken regarding the number of rules. As each rule represents a part of the knowledge presented, eliminating rules implies omitting information. The fuzzy controller is designed with the help of a control algorithm that considers fuzzification and defuzzification, as well as and all the rules all other rules pertaining to ****. Table 1 shows the base rules adopted for the current fuzzy control in a matrix format.

http://dx.doi.org/10.5772/59814

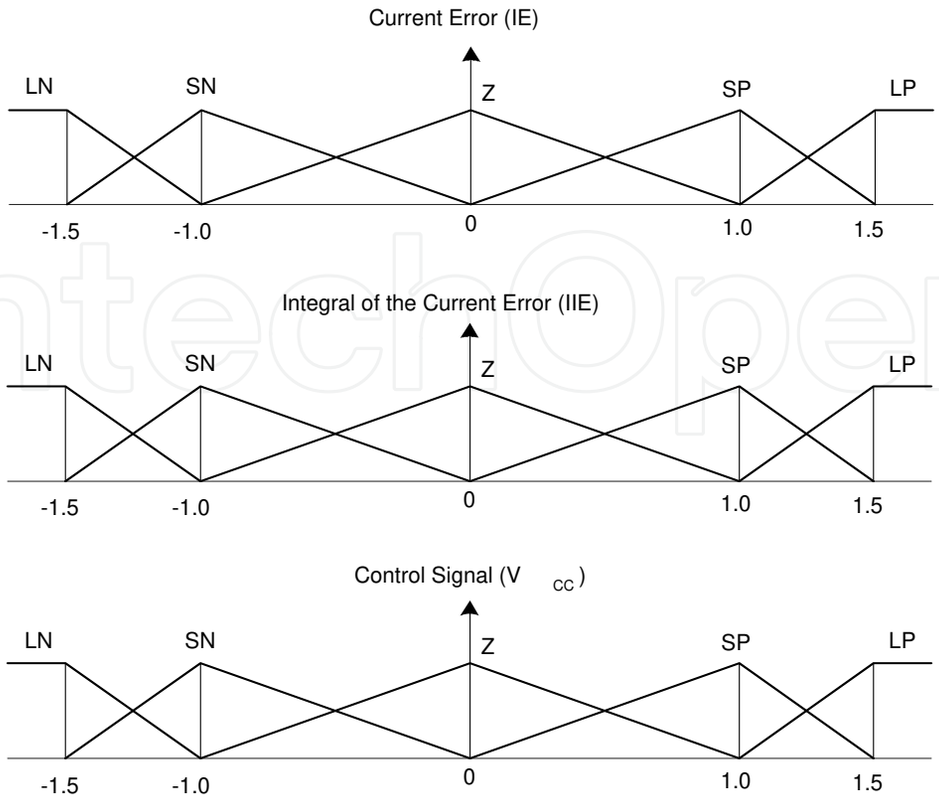


Figure 14. Representation of the pertinence functions.

		(IE)				
	(Vcc)	**LN**	**SN**	**ZE**	**SP**	**LP**
	LN	LN	LN	LN	SN	ZE
(IEE)	**SN**	LN	LN	SN	ZE	SP
	ZE	LN	SN	ZE	SP	LP
	SP	SN	ZE	SP	LP	LP
	LP	ZE	SP	LP	LP	LP

Where: LN = Large Negative; SN = Small Negative; ZE = Zero; SP = Small Positive; LP = Large Positive.

Table 1. Base rules for the current fuzzy control.

As illustrated in Figures 13 and 14, an input numerical value can be a member of more than one fuzzy set. For this, it is sufficient for the value to be situated in an overlapping region. So it means that, for a specific pair of values IE and IIE, more than one rule can be activated. Therefore, the existence of a means for combining the control actions activated by each rule is required so that a simple but significant action can be performed by combining all of these rules. This is the function of the defuzzifier. There are several methods for transformation of the output fuzzy set into a precise value, the latter representing the solution. The centroid method is well-indicated for control systems. Using this method, we can calculate the gravity

centre of the activated fuzzy rules, producing a result that is sensitive to all rules and which tends to dislocate itself smoothly through the control surface. The defuzzification by the centroid method selects the output as a value corresponding to the gravity centre of the output pertinence function as is given by (34) as:

$$X_0 = \frac{\int x\mu(x_1).\mu(x_2)dx}{\int \mu(x_1).\mu(x_2)dx} = \frac{\sum_{i=1}^{n} x_i \mu(IE_i).\mu(IIE_i)}{\sum_{i=1}^{n} \mu(IE_i).\mu(IIE_i)} \quad (34)$$

Where:

Vcc = X_0 = Gravity centre of the output pertinence function

x_i = Gravity centre of the output pertinence function activated by each rule

μ (x_i) = Pertinence function activated by each rule

7.4. Design of the fuzzy controller — Speed regulator

Similar to the current regulator, a diffuse speed regulator can be selected. The variables are the speed error, its integral and the regulator output information, which provides the input of the current regulator. The error of this grid will be the difference between the speed reference (n_{ref}) and the real speed (n_{real}). Corresponding pertinence functions are defined for the speed grid variables. The rules table can be similar to the current grid (Table 1), as it is a standard for several types of fuzzy controller applications. The scale factors (or gains) of a diffuse controller for the speed grid can be termed as GEn and GVn, respectively.

8. Experimental results

The data input parameters for digital control fuzzy regulators were as follows.

Speed regulator

Gen = 5.2 (proportional gain The practice adjusted value was 4.0. GVn = 2.17 (integral gain-inverse of the integral time constant τ_n); digital filter of the reference value – time constant = Tgs_1 = 460 [ms] (used only for digital control using conventional PI regulators).

Filter of the speed feedback transducer (analogical) – time constant = Tgn = 100 [ms] (implemented with RC components for both controls as digital fuzzy regulators and conventional ones).

Current regulator

GEi = 0.8 (proportional gain). The practice adjusted value was 0.1. GVi = 73.2 (integral gain-inverse of the integral time constant τ_i); digital filter of the reference value – time constant = Tgs2 = 15.84 [ms] (used only for digital control using conventional PI regulators).

http://dx.doi.org/10.5772/59814

Filter of the current feedback transducer (analogical) – time constant = Tgi = 1.5 [ms] (implemented with RC components, for both controls, as digital fuzzy regulators and conventional ones).

The dynamic behaviour of the controlled drive was verified. The current limiting was adjusted to 1.20 [pu]. The graphs of illustrations were provided with scales and locations: current (top)- 1 division = 5.5 [A]; speed (bottom)- 1 division = 1000[rpm]; time/div.= 2.5[s].

The machine data were: 1.7[kW], 220[V], 7.72 [A], 1500[rpm].

Figure 15 shows the machine speed and current response during start-up and for a load disturbance torque of 0.33 pu, negative and positive, respectively, using conventional digital PI regulators. The perfect current limiting and speed regulation could then be observed.

Figure 16 shows the machine current and speed response when the negative and positive steps of 0.33 pu in the speed reference set point occurred, respectively, using conventional digital PI regulators.

Figure 17 shows the response of the machine speed and current during start-up and for a load disturbance torque of 0.33 pu, negative and positive, respectively, using digital PI fuzzy regulators. The perfect current limiting and speed regulation could then be observed.

Figure 18 shows the machine current and speed response when negative and positive steps of 0.33 pu in the speed reference set point occurred, respectively, using digital PI fuzzy regulators.

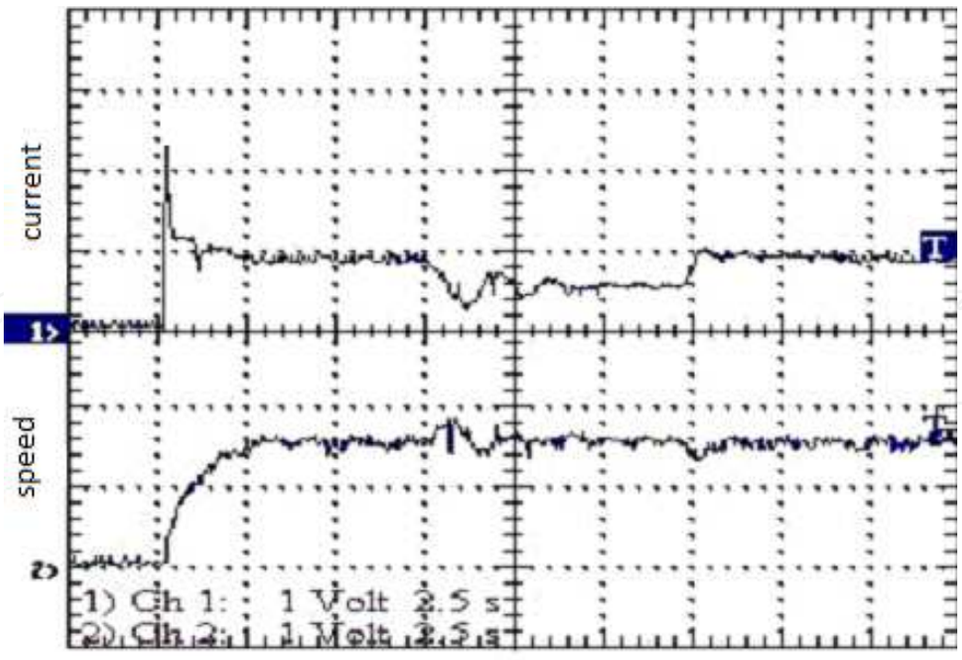


Figure 15. Machine current and speed during start-up and for a load disturbance torque of 0.33 pu, negative and positive, respectively, using conventional digital PI regulators.

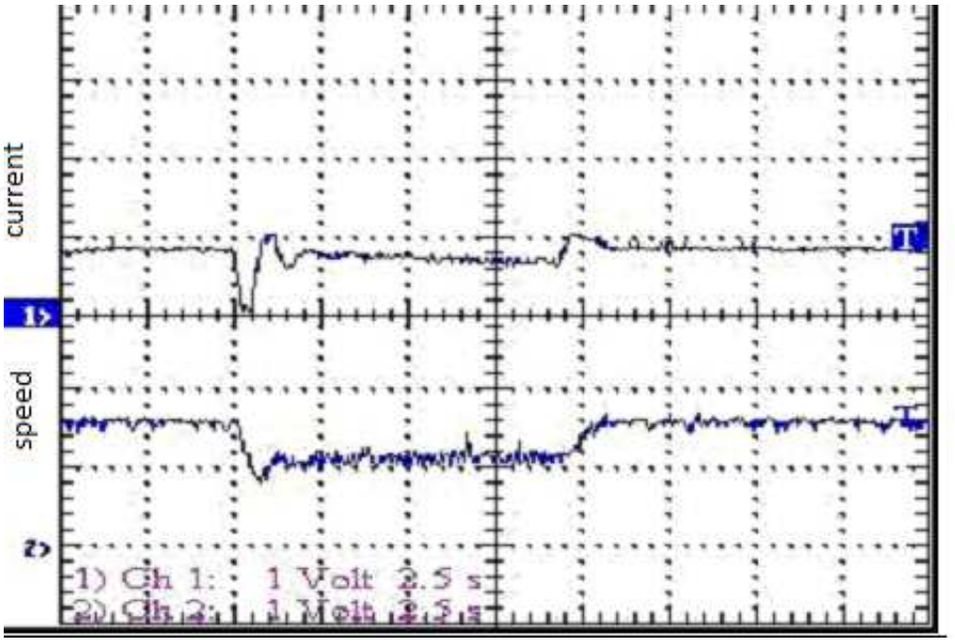


Figure 16. Machine current and speed when a negative and positive step of 0.33 pu in the speed reference set point occurred, respectively, using conventional digital PI regulators.

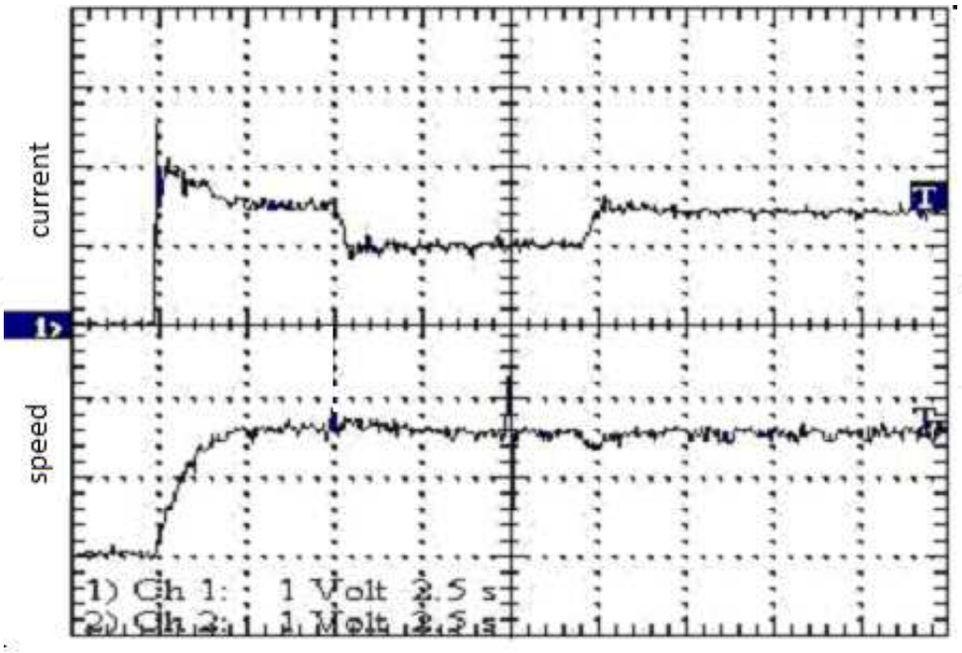


Figure 17. Machine current and speed during start-up and for a load disturbance torque of 0.33 pu, negative and positive, respectively, using digital PI fuzzy regulators.

http://dx.doi.org/10.5772/59814

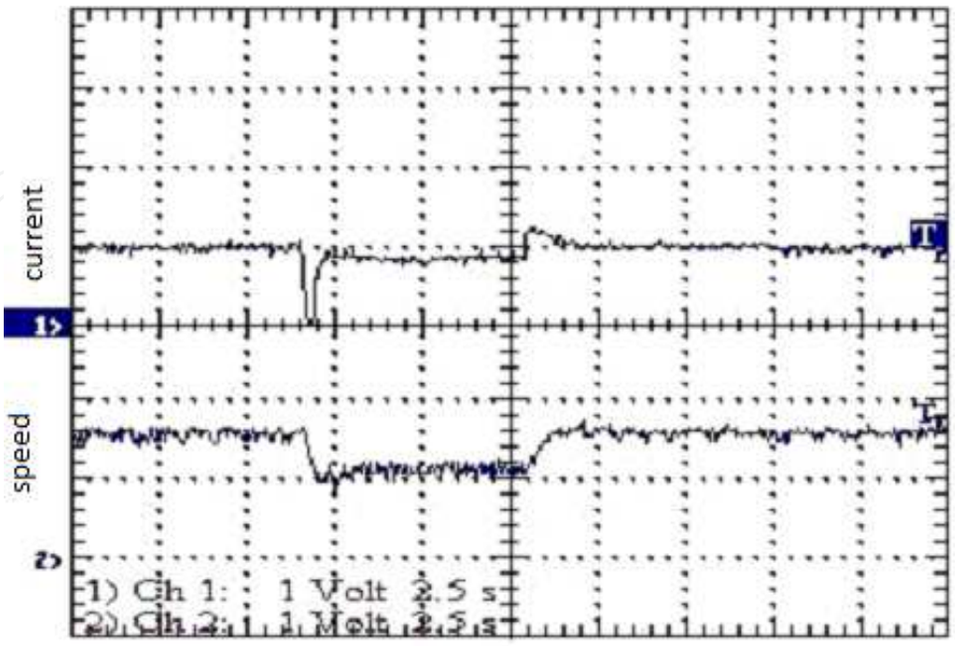


Figure 18. Machine current and speed when a negative and positive step of 0.33 pu in the speed reference set point occurred, respectively, using digital PI fuzzy regulators.

9. Conclusion

The experimental results obtained in the laboratory for the present study were satisfactory. The control with fuzzy regulators for the driving of a DC machine with series excitation was shown to be stable and efficient.

The most significant contribution of this work is the experimental implementation of fuzzy regulators in the control application of a non-linear DC series-motor drive. The system is simple to implement, for both DC motors and AC machines, replacing the traditional analogue controllers and allowing for an inexpensive and simple design. A comparison of dynamic drive performance showed that when conventional digital PI and fuzzy regulators were used, a response to steps in load torque (load disturbance torque) and in the speed reference set point was less oscillatory that when digital PI fuzzy regulators were employed, and comparatively so when digital conventional PI regulators were used. In fact, fuzzy logic control was extremely efficient for controlling non-linear systems [7].

Due to the wide flexibility offered by this type of control, the developed technique and the used equipment outlined in this paper can also be used to control other types of general systems.

Appendix

```
CONTROLED DC MOTOR DRIVE PROGRAM USING FUZZY LOGICS (PI-FUZZY) REGULATORS:
/*
*************************************************************************
*************
* Program    : PI Fuzzy Controller                    *
* Description    : Digital control actuator for a series DC motor using a  *
*  PCL-711B card     *
* Version    : 2                          *
* Date    : 25/07/2014                    *
*************************************************************************
*/
include <stdio.h>
#include <conio.h> /* Accept directives, including codes*/
#include <stdlib.h> /* of sources, of others */
#include <dos.h> /* programs and directories. */
#include <timer.h> /* program timer */
#include <math.h> /* math functions */
/* Global Variables Declaration */
extern "C" pcl711(int, unsigned int *); /* Includes Function "pcl711"
integer defined, unsigned in a separate module using "C" language*/
unsigned int param[60];    /* Definition of an array of data that make up the
table and unsigned integer parameters */
unsigned int datain[200], dataout[200]; /* 10 integer data Buffer +
                                for conversion*/
unsigned int far * datin, * datout;
/* Above data buffer address
                    - pointer - type integer and long:
                    2 words with range of 1Mbyte */
int tecla,i, cont1, cont2=0;      /* Keyboard reading variables
and number of channels */
/* Variables and control floating points */
float nRef= 1.00, nReal=0.0, ne=nRef, iRef, iR,ie, iReal=0.0 , ilim=1.2;
float a1, a2, b1, b2, b3, b4;
float DataBuf[3];
float Vc1, e, v , Vi1=0.0, j, dd1 ,u1 , Gv1 = 5.2, Ge1=2.17, dt=0.003;
float
ceNL=-1.2, ceNS=-1.0, ceZE= 0.0, cePS= 1.0, cePL= 1.2,
beNL= 1.0, beNS= 1.0, beZE= 1.0, bePS= 1.0, bePL= 1.0,
cvNL=-1.2, cvNS=-1.0, cvZE= 0.0, cvPS= 1.0, cvPL= 1.2,
bvNL= 1.0, bvNS= 1.0, bvZE= 1.0, bvPS= 1.0, bvPL= 1.0,
caNL=-1.2, caNS=-1.0, caZE= 0.0, caPS= 1.0, caPL= 1.2,
ueNL, ueNS, ueZE, uePS, uePL,
uvNL, uvNS, uvZE, uvPS, uvPL;
float r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16,
r17, r18, r19, r20, r21, r22, r23, r24, r25;
float Vc2,e2,v2, Vi2=0.0, dd2, Gv2 = 0.8 , Ge2 = 73.2 , u2 ;
float
ceNL2=-1.5, ceNS2=-1.0, ceZE2= 0.0, cePS2= 1.0, cePL2= 1.5,
beNL2= 1.0, beNS2= 1.0, beZE2= 1.0, bePS2= 1.0, bePL2= 1.0,
```

http://dx.doi.org/10.5772/59814

```
cvNL2=-1.5, cvNS2=-1.0, cvZE2= 0.0, cvPS2= 1.0, cvPL2= 1.5,
bvNL2= 1.0, bvNS2= 1.0, bvZE2= 1.0, bvPS2= 1.0, bvPL2= 1.0,
caNL2=-1.5, caNS2=-1.0, caZE2= 0.0, caPS2= 1.0, caPL2= 1.5,
ueNL2, ueNS2, ueZE2, uePS2, uePL2,
uvNL2, uvNS2, uvZE2, uvPS2, uvPL2;
float r26, r27, r28, r29, r30, r31, r32, r33, r34, r35, r36, r37, r38, r39,
r40, r41,
r42, r43, r44, r45, r46, r47, r48, r49, r50;
/* Variable Declaration void - means it does not return a value */
void conv_ad(void);
void conv_da(void);
void control1(void);
void control2(void);
void teclado(void);
/* Analog, Digital Conversion Subroutine */
void conv_ad()
{
unsigned int i;
/* Pointer - Memory space - Variable that contains an address, */
/* usually another variable's address."  */
datin = datain;  /* Assigns the equivalent value to the datin pointer... data-
in variable */
param[0] = 0;  /* Card number */
param[1] = 0x220; /* I/O Base Address */
/* sampling frequency = card base frequency /(C1 * C2) */
/* 2M / (10 * 10) = 20 kHz */
param[5] = 10;  /* Constant divisor C1 */
param[6] = 10;  /* Constant divisor C2 */
param[7] = 0;  /* Mode Trigger, 0 : pacer trigger
Allows D/I functions    */
/* Buffer Offset, memory address (Buffer) where data will be stored, Segment,
data Buffer length
*/
param[10] = FP_OFF(datin); /* A/D Buffer A Offset */
param[11] = FP_SEG(datin); /* A/D Buffer A Segment */
param[12] = 0;  /* Buffer B Addresss (unused)*/
param[13] = 0;  /* Segment- Unused, set to 0 */
/* The A/D conversion covers two input channels, channel 1 - current, and chan-
nel 0 - speed, with values in pu adjusted in +/- 5 V */
param[14] = 2;  /* Number of A/D conversions */
param[15] = 0;  /* Channel of the A/D conversion initiation*/
param[16] = 1;  /* Stop A/D conversion channel*/
param[17] = 0;  /* Channel gains, 0 : +/- 5V */
/* conversion A/D fault indication*/
pcl711(3, param); /* Function 3 : Hardware initialization */
if (param[45] != 0) { /* If parameter 45 is different from 0, do: */
    clrscr(); /* Clear screen  */
    printf("\n DRIVER INITIALIZATION FAILED!"); /* Print */
    getch(); /* Shows exit screen  */
    exit(1); /* Closes loop and exit with status 1 - Error */
        }
pcl711(4, param); /* Function 4 : Conversor A/D initialization*/
```

```
if (param[45] != 0) {
    clrscr();
    printf("\n A/D INITIALIZATION FAILURE!");
    getch();
    exit(1);
         }
pcl711(5, param); /* Function 5 : verification of the number of A/D conver-
sions */
if (param[45] != 0) {
    clrscr();
    printf("\n A/D DATA TRANSFER SOFTWARE FAILURE!");
    getch();
    exit(1);
        }
/* A/D Conversions */
for (i = 0; i < param[14]; i++) /* Sampled data - channels 0 e 1 */
    {
    DataBuf[i] = datain[i] & 0xFFF;
        /* Data collect for the buffer on the address 0xFFF
     (the first three hexadecimal digits can be reset because the
remaining, are sufficient to support 4096 binary digits) */
    DataBuf[i] =((5.0 - (-5)) * DataBuf[i] / 4096) + (-5);
      /* conversion so that the voltage signal is available for the recursive
control equations
     (5 - (-5)) : Input range A/D (-5V to 5V)
     4096 : Scale range of the A/D - 12 bit
    DataBuf : Input data of the A/D
     (-5) : Beginning of the scale of the A/D "-5" V
     */
     }
/* Feedback voltage reading for the speed and current loops, under nominal
load and speed.*/
/* speed signal conversion - correction to pu*/
    nReal=(DataBuf[0]/1.500);
/* current signal conversion - correction to pu */
iReal=(DataBuf[1]/1.493);
}
void control1()
{
ne = nRef - nReal ; /* Erro */
Vi1 = Vi1 + Gv1*ne*dt; /* Error integral */
if (Vi1 <cvNL) Vi1 = cvNL ; /* Limits */
if (Vi1 >cvPL) Vi1 = cvPL ;
e = Ge1 * ne ;
v = Vi1 ;
teclado(); /* Performs the subroutine that inspects the keystroke */
/* Fuzzification : Triangular Functions */
/* Analyzing with respect to "e" */
if ( ( ( ceNL - beNL ) <= e ) && ( e <= ( ceNL + beNL ) ) )
ueNL = 1.0 - ( fabs( ceNL - e) ) / beNL ;
elseueNL = 0.0 ;
if ( e <ceNL )
```

http://dx.doi.org/10.5772/59814

```
ueNL = 1.0 ;
if ( ( ( ceNS – beNS ) <= e ) && ( e <= ( ceNS + beNS ) ) )
ueNS = 1.0 – ( fabs( ceNS – e ) ) / beNS ;
elseueNS = 0.0 ;
if ( ( ( ceZE – beZE ) <= e ) && ( e <= ( ceZE + beZE ) ) )
ueZE = 1.0 – ( fabs( ceZE – e ) ) / beZE ;
elseueZE = 0.0 ;
if ( ( ( cePS – bePS ) <= e ) && ( e <= ( cePS + bePS ) ) )
uePS = 1.0 – ( fabs( cePS – e ) ) / bePS ;
elseuePS = 0.0 ;
if ( ( ( cePL – bePL ) <= e ) && ( e <= ( cePL + bePL ) ) )
uePL = 1.0 – ( fabs( cePL – e ) ) / bePL ;
elseuePL = 0.0 ;
if ( e >cePL)
uePL = 1.0 ;
/* Analyzing with respect to "v" */
if ( ( ( cvNL – bvNL ) <= v ) && ( v <= ( cvNL + bvNL ) ) )
uvNL = 1.0 – ( fabs( cvNL – v ) ) / bvNL ;
elseuvNL = 0.0 ;
if ( v <cvNL)
uvNL = 1.0 ;
if ( ( ( cvNS – bvNS ) <= v ) && ( v <= ( cvNS + bvNS ) ) )
uvNS = 1.0 – ( fabs( cvNS – v ) ) / bvNS ;
elseuvNS = 0.0 ;
if ( ( ( cvZE – bvZE ) <= v ) && ( v <= ( cvZE + bvZE ) ) )
uvZE = 1.0 – ( fabs( cvZE – v ) ) / bvZE ;
elseuvZE = 0.0 ;
if ( ( ( cvPS – bvPS ) <= v ) && ( v <= ( cvPS + bvPS ) ) )
uvPS = 1.0 – ( fabs( cvPS – v ) ) / bvPS ;
elseuvPS = 0.0 ;
if ( ( ( cvPL – bvPL ) <= v ) && ( v <= ( cvPL + bvPL ) ) )
uvPL = 1.0 – ( fabs( cvPL – v ) ) / bvPL ;
elseuvPL = 0.0 ;
if ( v >cvPL )
uvPL = 1.0 ;
/* Logical Implications – Product Operator*/
r1 = ueNL * uvNL ;
r2 = ueNL * uvNS ;
r3 = ueNL * uvZE ;
r4 = ueNL * uvPS ;
r5 = ueNL * uvPL ;
r6 = ueNS * uvNL ;
r7 = ueNS * uvNS ;
r8 = ueNS * uvZE ;
r9 = ueNS * uvPS ;
r10 = ueNS * uvPL ;
r11 = ueZE * uvNL ;
r12 = ueZE * uvNS ;
r13 = ueZE * uvZE ;
r14 = ueZE * uvPS ;
r15 = ueZE * uvPL ;
r16 = uePS * uvNL ;
```

```
r17 = uePS * uvNS ;
r18 = uePS * uvZE ;
r19 = uePS * uvPS ;
r20 = uePS * uvPL ;
r21 = uePL * uvNL ;
r22 = uePL * uvNS ;
r23 = uePL * uvZE ;
r24 = uePL * uvPS ;
r25 = uePL * uvPL ;
/* Defuzzification */
dd1 =
caNL*r1+caNL*r2+caNL*r3+caNS*r4+caZE*r5+caNL*r6+caNL*r7+caNS*r8+
caZE*r9+caPS*r10+caNL*r11+caNS*r12+caZE*r13+caPS*r14+caPL*r15+
caNS*r16+caZE*r17+caPS*r18+caPL*r19+caPL*r20+caZE*r21+caPS*r22+
caPL*r23+caPL*r24+caPL*r25;
u1 = dd1 /
(r1+r2+r3+r4+r5+r6+r7+r8+r9+r10+r11+r12+r13+r14+r15+r16+r17+r18+r19+
r20+r21+r22+r23+r24+r25);
//u = u/10.0 ;
if (u1 < -1.0) u1 = -ilim;
if (u1 > 1.0 ) u1 = ilim;
}
void control2()
{
/* Current Loop */
if(iRef>ilim) iRef = ilim ;
if(iRef<-ilim) iRef =-ilim ;
ie = iReal - iRef ; /* Error */
Vi2 = Vi2 + Gv2*ie*dt; /* Integral Error */
if (Vi2 < cvNL2) Vi2 = cvNL2 ; /* Limits */
if (Vi2 > cvPL2) Vi2 = cvPL2 ;
e2 = Ge2 * ie ;
v2 = Vi2 ;
/*: Triangular Functions Fuzzification */
/* Analysis in Relation to "e2" */
if ( ( ( ceNL2 - beNL2 ) <= e2 ) && ( e2 <= ( ceNL2 + beNL2 ) ) )
ueNL2 = 1.0 - ( fabs( ceNL2 - e2) ) / beNL2 ;
else ueNL2 = 0.0 ;
if ( e2 < ceNL2 )
ueNL2 = 1.0 ;
if ( ( ( ceNS2 - beNS2 ) <= e2 ) && ( e2 <= ( ceNS2 + beNS2 ) ) )
ueNS2 = 1.0 - ( fabs( ceNS2 - e2 ) ) / beNS2 ;
else ueNS2 = 0.0 ;
if ( ( ( ceZE2 - beZE2 ) <= e2 ) && ( e2 <= ( ceZE2 + beZE2 ) ) )
ueZE2 = 1.0 - ( fabs( ceZE2 - e2 ) ) / beZE2 ;
else ueZE2 = 0.0 ;
if ( ( ( cePS2 - bePS2 ) <= e2 ) && ( e2 <= ( cePS2 + bePS2 ) ) )
uePS2 = 1.0 - ( fabs( cePS2 - e2 ) ) / bePS2 ;
else uePS2 = 0.0 ;
if ( ( ( cePL2 - bePL2 ) <= e2 ) && ( e2 <= ( cePL2 + bePL2 ) ) )
uePL2 = 1.0 - ( fabs( cePL2 - e2 ) ) / bePL2 ;
else uePL2 = 0.0 ;
```

http://dx.doi.org/10.5772/59814

```
if ( e2 > cePL2)
uePL2 = 1.0 ;
/* Analysis in Relation to "v2" */
if ( ( ( cvNL2 - bvNL2 ) <= v2 ) && ( v2 <= ( cvNL2 + bvNL2 ) ) )
uvNL2 = 1.0 - ( fabs( cvNL2 - v2 ) ) / bvNL2 ;
else uvNL2 = 0.0 ;
if ( v2 < cvNL2)
uvNL2 = 1.0 ;
if ( ( ( cvNS2 - bvNS2 ) <= v2 ) && ( v2 <= ( cvNS2 + bvNS2 ) ) )
uvNS2 = 1.0 - ( fabs( cvNS2 - v2 ) ) / bvNS2 ;
else uvNS2 = 0.0 ;
if ( ( ( cvZE2 - bvZE2 ) <= v2 ) && ( v2 <= ( cvZE2 + bvZE2 ) ) )
uvZE2 = 1.0 - ( fabs( cvZE2 - v2 ) ) / bvZE2 ;
else uvZE2 = 0.0 ;
if ( ( ( cvPS2 - bvPS2 ) <= v2 ) && ( v2 <= ( cvPS2 + bvPS2 ) ) )
uvPS2 = 1.0 - ( fabs( cvPS2 - v2 ) ) / bvPS2 ;
else uvPS2 = 0.0 ;
if ( ( ( cvPL2 - bvPL2 ) <= v2 ) && ( v2 <= ( cvPL2 + bvPL2 ) ) )
uvPL2 = 1.0 - ( fabs( cvPL2 - v2 ) ) / bvPL2 ;
else uvPL2 = 0.0 ;
if ( v2 > cvPL2 )
uvPL2 = 1.0 ;
/* Logical Implications - Product Operator */
r26 = ueNL2 * uvNL2 ;
r27 = ueNL2 * uvNS2 ;
r28 = ueNL2 * uvZE2 ;
r29 = ueNL2 * uvPS2 ;
r30 = ueNL2 * uvPL2 ;
r31 = ueNS2 * uvNL2 ;
r32 = ueNS2 * uvNS2 ;
r33 = ueNS2 * uvZE2 ;
r34 = ueNS2 * uvPS2 ;
r35 = ueNS2 * uvPL2 ;
r36 = ueZE2 * uvNL2 ;
r37 = ueZE2 * uvNS2 ;
r38 = ueZE2 * uvZE2 ;
r39 = ueZE2 * uvPS2 ;
r40 = ueZE2 * uvPL2 ;
r41 = uePS2 * uvNL2 ;
r42 = uePS2 * uvNS2 ;
r43 = uePS2 * uvZE2 ;
r44 = uePS2 * uvPS2 ;
r45 = uePS2 * uvPL2 ;
r46 = uePL2 * uvNL2 ;
r47 = uePL2 * uvNS2 ;
r48 = uePL2 * uvZE2 ;
r49 = uePL2 * uvPS2 ;
r50 = uePL2 * uvPL2 ;
/* Defuzzification */
dd2 =
caNL2*r26+caNL2*r27+caNL2*r28+caNS2*r29+caZE2*r30+caNL2*r31+caNL2*r32+caNS2*r33
+
```

```
caZE2*r34+caPS2*r35+caNL2*r36+caNS2*r37+caZE2*r38+caPS2*r39+caPL2*r40+
caNS2*r41+caZE2*r42+caPS2*r43+caPL2*r44+caPL2*r45+caZE2*r46+caPS2*r47+
caPL2*r48+caPL2*r49+caPL2*r50;
u2 = dd2 /
(r26+r27+r28+r29+r30+r31+r32+r33+r34+r35+r36+r37+r38+r39+r40+r41+r42+r43+r44+
r45+r46+r47+r48+r49+r50);
if (u2 < 0.0) u2 = 0.0;
if (u2 > 1.0) u2 = 1.0;
}
voidteclado()
{
/* The activation of these subroutine keys, allows online adjustment
system parameters, where settings are displayed on the output */
/* Keys "k" and "l" operating in the value of the reference speed */
if (tecla==108 &&nRef<1.0) nRef=nRef+0.01; /* Limited adjustment to */
if (tecla==107 &&nRef>-1.0) nRef=nRef-0.01; /* range -1.0<nRef<1.0 */
/* Keys "o" and "p" operating in the value of the reference speed */
if (tecla==112 &&nRef<1.0) nRef=nRef+0.05; /* Limited adjustment to */
if (tecla==111 &&nRef>-1.0) nRef=nRef-0.05; /* range -1.0<nRef<1.0 */
/* Keys "q" e "w" operating on the gain Gv1 */
if (tecla==119 && Gv1<99.95) Gv1 = Gv1+0.05 ;
if (tecla==113 && Gv1>0.05) Gv1 = Gv1-0.05 ;
/* Keys "s" e "d" operating on the gain Ge1 */
if (tecla==100 && Ge1< 99.95) Ge1=Ge1+0.05;
if (tecla==115 && Ge1> 0.05) Ge1=Ge1-0.05;
/* Keys "g" e "h" operating on the gain Gv2 */
if (tecla==103 && Gv2< 9.95) Gv2=Gv2+0.05;
if (tecla==104 && Gv2< 0.05) Gv2=Gv2-0.05;
/* Keys "m" e "n" operating on the gain Ge2 */
if (tecla==109 && Ge2< 9.95) Ge2=Ge2+0.05;
if (tecla==110 && Ge2> 0.05) Ge2=Ge2-0.05;
}
/* Digital-Analogic Conversion Subroutine */
voidconv_da()
{
datout=dataout; /* Assigns the equivalent value to the datout pointer
... dataout variable */
param[0]=0; /* Card number  */
param[1]=0x220; /* I/O base address */
/* Buffer Offset , the memory address (Buffer) where data
will be stored. Segment , the length of the data buffer */
param[20] = FP_OFF(datout); /* Buffer A Offset D/A output data */
param[21] = FP_SEG(datout); /* Buffer A segment D/A output data */
param[22] = 0; /* Buffer B output address(unused) */
param[23] = 0; /* Output segment- Unused, set 0 */
param[24] = 1; /* Number of D/A conversions*/
param[25] = 0; /* D/A conversion initialization channel */
param[26] = 0; /* D/A conversion stop channel */
/* D/A conversion fault indicator */
pcl711(3, param); /* Function 3 : Hardware initialization */
if (param[45] != 0) { /* If parameter 45 different from 0, do: */
    clrscr(); /* Clear screen  */
```

http://dx.doi.org/10.5772/59814

```
    printf("\n DRIVER INITIALIZATION FAILURE!"); /* Print */
    getch(); /* Output screen display  */
    exit(1); /* Close loop and exit with status 1 – Error */
         }
pcl711(12, param); /* Function 12: D/A conversor initialization */
if (param[45] != 0) {
    clrscr();
    printf("\n D/A INITIALIZATION FAILURE !");
    getch();
    exit(1);
        }
pcl711(13, param); /* Function 13: number of D/A conversions verification*/
if (param[45] != 0) {
    clrscr();
    printf("\n D/A DATA TRANSFER SOFTWARE FAILURE!");
    getch();
    exit(1);
     }
/* conversion for the output voltage signal of the recursive control equations
in pu to occupy an address space of the output data buffer.  */
dataout[0]=(4095*u2); /* Base pu = 4095 */
}
/* Main program */
void main(void)
{
Timer t;  /* Timer"t" */
clrscr();  /* Clear screen */
delay(500);
textbackground(4); /* Defines Red Background (4) */
gotoxy(5,2);
cprintf("FUZZY CONTROLLER FOR A DC MOTOR SPEED CONTROL: );
gotoxy(5,4);
delay(500);
cprintf(" Student : Otavio Henrique Salvi Vicentini n 9025 ");
gotoxy(5,5);
cprintf(" Advisor : prof. Angelo J. J. Rezek Date : 25/09/2014 ");
delay(500);
textbackground(1); /* Defines a Blue Background (1) */
gotoxy(1,15); cprintf(" [O] – Decreases the Speed [0.05] ");
gotoxy(40,15); cprintf(" [P] – Increases the Speed [0.05] ");
gotoxy(1,16); cprintf(" [K] – Decreases the Speed [0.01] ");
gotoxy(40,16); cprintf(" [L] – Increases the Speed [0.01] ");
gotoxy(1,17); cprintf(" [Q] – Decreases the Gain Gv1 [0.05] ");
gotoxy(40,17); cprintf(" [W] – Increases the Gain Gv2 [0.05] ");
gotoxy(1,18); cprintf(" [S] – Decreases the Gain Ge1 [0.05] ");
gotoxy(40,18); cprintf(" [D] – Increases the Gain Ge1 [0.05] ");
gotoxy(1,19); cprintf(" [G] – Decreases the Gain Gv2 [0.05] ");
gotoxy(40,19); cprintf(" [H] – Increases the Gain Gv2 [0.05] ");
gotoxy(1,20); cprintf(" [M] – Decreases the Gain Ge2 [0.05] ");
gotoxy(40,20); cprintf(" [N] – Increases the Gain Ge2 [0.05] ");
textbackground(4); /* Defines Red Background (4) */
gotoxy(13,24); cprintf(" PRESS ESC TO END PROGRAM ");
```

```
do
{
tecla=0;
if (kbhit()) tecla=getch(); /* Control by keyboard */
j++;  /* Increases the number of interactions by 1 */
t.reset();  /* Resets the timer "t" */
t.start();  /* Starts the timer */
asm cli;
conv_ad();  /* Performs A/D subroutine conversion */
asmsti;
ne = nRef-nReal;
control1();
if(iRef<ilim&&iRef>-ilim) control1();
if(iRef==ilim&& ne<0.0) control1();
if(iRef==-ilim&& ne>0.0) control1();
iRef = u1;
ie=iReal-iRef;
if(u2<0.9 && u2>.10) control2();
if(u2>=0.9 &&ie<0.0) control2();
if(u2<=.10 &&ie>0.0) control2();
asm cli;
conv_da(); /* Performs A/D subroutine conversion */
asmsti;
if (j>=100) /* Displays instant results each 100 interactions in the screen */
{
j=0;
gotoxy(7,8);
textbackground(1);
cprintf(" Time spent : %1.6f (s)", dt);
gotoxy(7,9);
cprintf(" Channel [0] - Speed : % 1.3f (pu) Gv1 %1.3f Ge1 %1.3f " ,
nReal,Gv1,Ge1);
gotoxy(7,10);
cprintf(" Channel [1] - Current : % 1.3f (pu) Gv2 %1.3f Ge2 %1.3f " , iRe-
al,Gv2,Ge2);
gotoxy(7,11);
cprintf(" Reference Speed : %1.2f (pu) iRef : %1.2f (pu) " , nRef,iRef);
gotoxy(7,12);
cprintf(" ne : %1.2f ie : %1.2f Vcontr : %1.2f (pu) ", ne, ie, u2);
}
t.stop(); /* Stops timer */
dt = t.time(); /* Sampling time */
} while (tecla!=27); /* Program ends when the Esc key and is pressed.*/
}
CONTROLED DC MOTOR DRIVE PROGRAM USING PI (PROPORTIONAL-INTEGRAL) REGULATORS,
USING RECURSIVE EQUATIONS DERIVED FROM Z TRANSFORM:
/*
*************************************************************************
******************
*     Program    : MCC Control                                *
*     Description    : Digital control for a series DC motor using convention-
al digital
```

http://dx.doi.org/10.5772/59814

```
* PI regulators       *
*   PCL-711B board         *
*     Version         : 2                              *
*     Date        : 29/09/2014                              *
**************************************************************************
********************
*/
/* Inclusão de Diretivas */
#include <stdio.h>
#include <conio.h>        /* Accept directives, including codes */
#include <stdlib.h>       /* from other sources */
#include <dos.h>         /* programs and folders */
#include <timer.h>
/* Declaração de Variáveis Globais */
extern "C" pcl711(int, unsigned int *);      /*Includes function "pcl711" inte-
ger defined, unsigned in module separated using language "C"*/
unsigned int param[60];             /* Defining an data array -
- which makes up an unsigned integer parameter table */
unsigned int datain[200], dataout[200];     /* 10 integer data buffer +
                           for conversion*/
unsigned int far * datin, * datout;         /* Buffer address of data above
                        - pointer - long and integer type:
                        2 words with 1Mbyte range */
int tecla,i;                        /* Keyboard reading variables
                           and number of channels*/
/* Control floating point variables */
float nRef=0.72, nReal=0.0, ne=nRef;
float ne_1=0.1, nReal_1=0.0;
float iRef=1.20, iR=1.20, ie=-1.20, iReal=0.0, Ilim=1.20, iRefer=1.20;
float iRef_1=1.20, iR_1=1.20, ie_1=-1.20, iReal_1=0.0, iRefer_1=1.20;
float Tn=0.460, Tgs2=0.01584, Ti=0.01366;
float VRn=5.20, VRi=0.8, Vcc=0.1, Vcc_1=0.1, Vcontr=0.1;
float a1, a2, b1, b2, b3, b4, T=0.003;
float DataBuf[3];
/* Variable declaration void - means it does not return a value */
void conv_ad(void);
void conv_da(void);
void control(void);
void control1(void);
void control2(void);
void teclado(void);
/* AD conversion - Parameters table */
void conv_ad()
{
unsigned int i;
    /* Pointer - Memory space - Variable that contains an address,
usually another variable's address */
datin = datain;  /* Assigns to the datin pointer the
                 equivalent value of datain variable */
param[0] = 0;  /* Board number */
param[1] = 0x220; /* I/O base address */
/* Sampling frequency = Board base frequency/(C1 * C2) */
```

```
            /* 2M / (10 * 10) = 20 KHz */
param[5] = 10;  /* Constant divisor pacer C1 */
param[6] = 10;  /* Constant divisor pacer C2 */
param[7] = 0;  /* Trigger Mode, 0 : pacer trigger
                    Allows D/I functions    */
/* Buffer offset , the memory address (buffer) where the data
will be stored. Segment, the length of the data buffer */
param[10] = FP_OFF(datin);      /* A/D Buffer A offset */
param[11] = FP_SEG(datin);      /* A/D Buffer A segment */
param[12] = 0;          /* Buffer B address (unused)*/
param[13] = 0;          /* Segment - unused, set to 0 */
/* The A/D conversion covers two input channels, channel 1 - current, and chan-
nel 0 - speed, with values in pu adjusted in +/- 5 V */
param[14] = 2;  /* Number of A/D conversions */
param[15] = 0;  /* Channel of the A/D conversion initiation*/
param[16] = 1;  /* Stop A/D conversion channel*/
param[17] = 0;  /* Channel gains, 0 : +/- 5V */
/* Conversion A/D fault indication*/
pcl711(3, param); /* Function 3 : Hardware initialization */
if (param[45] != 0) { /* If parameter 45 is different from 0, do: */
     clrscr(); /* Clear screen  */
     printf("\n DRIVER INITIALIZATION FAILED!"); /* Print */
     getch(); /* Shows exit screen  */
     exit(1); /* Closes loop e exit with status 1 - Error */
         }
pcl711(4, param);      /* Function 4 : Converter A/D initialization*/
if (param[45] != 0) {
     clrscr();
     printf("\n A/D INITIALIZATION FAILED!");
     getch();
     exit(1);
         }
pcl711(5, param);      /* Function 5 : Number of conversions A/D verification*/
if (param[45] != 0) {
     clrscr();
     printf("\n A/D DATA TRANSFER SOFTWARE FAILURE!");
     getch();
     exit(1);
         }
/* A/D conversions */
for (i = 0; i < param[14]; i++) /* Sampled data - channels 0 e 1 */
     {
     DataBuf[i] = datain[i] & 0xFFF;
        /* Data collect for the buffer on the address 0xFFF
     (the first three hexadecimal digits can be reset because the
remaining, sufficient to support 4096 binary digits) */
     DataBuf[i] =((5.0 - (-5)) * DataBuf[i] / 4096) + (-5);
      /* conversion so that the voltage signal is available for the recursive
equations control
     (5 - (-5)) : Input range A/D (-5V to 5V)
     4096 : Scale range of the A/D - 12 bit
     DataBuf : Input data of the A/D
```

http://dx.doi.org/10.5772/59814

```
     (-5) : Beginning of the scale of the A/D "-5" V
     */
     }
/* Feedback voltage reading for the speed and current loop mesh, under nominal
load and speed.*/
/* speed signal conversion - correction to pu*/
    nReal=(DataBuf[0]/1.500);
/* Current signal conversion - correction to pu */
    iReal=(DataBuf[1]/1.493);
}
/* Control */
/* Recursive Equations to perform functions of control */
void control()
{                    /* Sampling time considered T */
a1=T/((2*Tgs2)+T);
                    /*Tgs2= Filter time constant */
a2=((2*Tgs2)-T)/(T+(2*Tgs2));
b1=VRn+((VRn*T)/(2*Tn));        /* VRn= Speed regulator gain */
                    /* Tn= Speed regulator time constant */
b2=((VRn*T)/(2*Tn))-VRn;
b3=VRi+((VRi*T)/(2*Ti));        /* VRi= Current regulator gain */
                    /* Ti= Current regulator time constant */
b4=((VRi*T)/(2*Ti))-VRi;
}
/* Speed Regulator */
void control1()
    {
    iRef=(b1*ne)+(b2*ne_1)+iRef_1;    /* nreal= Speed feedback */
    iRef_1=iRef;                /* nRef= Post filter reference speed */
    ne_1=ne;                /* ne= Speed error */
                        /* iRef= Current reference - Output
                         speed regulator */
/* Reference Current Filter */
iRefer=iRef;
    iR=a1*(iRefer+iRefer_1)+a2*iR_1;    /* Which:   */
    iR_1=iR;                /* iR= Post filter current reference/
    iRefer_1=iRefer;            /* iRef= Reference current */
    if(iR>=Ilim) iR=Ilim;
    if(iR<=-Ilim) iR=-Ilim;          /* Current reference limit */
}
/* Current Regulator */
void control2()
    {                    /* Which:             */
    Vcc=(b3*ie)+(b4*ie_1)+Vcc_1;        /* ireal= feedback current */
    Vcc_1=Vcc;                /* iR= Post filter reference current */
    ie_1=ie;                /* ie= Current Error- In. regulator */
    if(Vcc>0.95) Vcontr=0.95;
    if(Vcc<.05) Vcontr=.05;            /* Control voltage limit */
    if(Vcc<=0.95 && Vcc>=.05) Vcontr=Vcc;
}
/* System parameters alteration */
void teclado()
```

```
{
/* The activation of these subroutine keys, allows online adjustment of system
parameters, with the settings being shown in the output screen */
     /* Keys "s" and "d" operating in the value of the reference speed */
    if (tecla==115 && nRef<1.0) nRef=nRef+1.44; /* Adjustment limited to */
    if (tecla==100 && nRef>-1.0) nRef=nRef-1.44; /* range -1.0<nRef<1.0 */
     /* Keys "k" and "l" operating in value of the reference speed */
    if (tecla==107 && nRef<1.0) nRef=nRef+0.01; /* Adjustment limited to */
    if (tecla==108 && nRef>-1.0) nRef=nRef-0.01; /* range -1.0<nRef<1.0 */
     /* Keys "o" and "p" operating in value of the reference speed */
    if (tecla==111 && nRef<1.0) nRef=nRef+0.05; /* Adjustment limited to */
    if (tecla==112 && nRef>-1.0) nRef=nRef-0.05; /* range -1.0<nRef<1.0 */
     /* Keys "f" and "v" operating in VRn gain */
    if (tecla==102 && VRn<40.0) VRn=VRn+0.05;
    if (tecla==118 && VRn>0.02) VRn=VRn-0.05;
     /* Keys "g" and "b" operating on time constant Tn */
    if (tecla==103 && Tn<5.00) Tn=Tn+0.05;
    if (tecla==98 && Tn>0.06) Tn=Tn-0.05;
/* Keys "h" and "n" operating in VRi gain */
    if (tecla==104 && VRi<10.0) VRi=VRi+0.05;
    if (tecla==110 && VRi>0.01) VRi=VRi-0.05;
     /* Keys "j" and "m" operating on time constant Ti */
    if (tecla==106 && Ti<1.000) Ti=Ti+0.005;
    if (tecla==109 && Ti>0.006) Ti=Ti-0.005;
}
/* D/A conversion - Parameter table */
void conv_da()
{
datout=dataout;        /* Assigns the equivalent value to the datout pointer */
param[0]=0; /* Card number  */
param[1]=0x220; /* I/O base address */
     /* Buffer Offset , the memory address (Buffer) where data
will be stored. Segment , the length of the data buffer */
param[20] = FP_OFF(datout); /* Buffer A Offset D/A output data */
param[21] = FP_SEG(datout); /* Buffer A segment D/A output data */
param[22] = 0; /* Buffer B output address(unused) */
param[23] = 0; /* Output segment- Unused, set 0 */
param[24] = 1; /* Number of D/A conversions*/
param[25] = 0; /* D/A conversion initialization channel */
param[26] = 0; /* D/A conversion stop channel */
/* D/A conversion fault indicator */
pcl711(3, param); /* Function 3 : Hardware initialization */
if (param[45] != 0) { /* If parameter 45 different from 0, do: */
     clrscr(); /* Clear screen  */
     printf("\n DRIVER INITIALIZATION FAILURE!"); /* Print */
     getch(); /* Output screen display  */
     exit(1); /* Close loop and exit with status 1 - Error */
         }
pcl711(12, param); /* Function 12: D/A converter initialization */
if (param[45] != 0) {
     clrscr();
     printf("\n D/A INITIALIZATION FAILURE !");
```

http://dx.doi.org/10.5772/59814

```
     getch();
     exit(1);
         }
pcl711(13, param); /* Function 13: number of D/A conversions verification*/
if (param[45] != 0) {
     clrscr();
     printf("\n D/A DATA TRANSFER SOFTWARE FAILURE!");
     getch();
     exit(1);
         }
     /* Conversion for the output voltage signal of the recursive control equa-
tions in pu to  . occupy an address space of the output data buffer. */
dataout[0]=(4095*Vcontr);
}
/* Main program */
void main(void)        /* Ensures that global variables do not return values */
{            /* Variable declarations */
int xx=0;        /* Initializes the number of interactions in 0*/
Timer t;            /* Time counter "t" */
clrscr();            /* Clear screen */
gotoxy(1,10); cprintf("[S] - Invert vel. [+]");
gotoxy(40,10); cprintf("[D] - Invert vel. [-]");
gotoxy(1,11); cprintf("[O] - Increments vel. [0.05]");
gotoxy(40,11); cprintf("[P] - Decrements vel. [0.05]");
gotoxy(1,12); cprintf("[K] - Increments vel. [0.01]");
gotoxy(40,12); cprintf("[L] - Decrements vel. [0.01]");
do{
tecla=0;
if (kbhit()) tecla=getch();
xx++;     /* Increments 1 to the number of interactions */
t.reset();     /* Resets timer "t" */
t.start();     /* Starts timing */
asm cli;
conv_ad();     /* Performs the A/ D conversion subroutine */
asm sti;
control();
teclado();     /* Performs the inspecting keystroke subroutine */
ne=nRef-nReal;
if(iR<Ilim && iR>-Ilim) control1();
if(iR==Ilim && ne<0.0) control1();
if(iR==-Ilim && ne>0.0) control1();
ie=iReal-iR;
if(Vcc<0.95 && Vcc>.05)control2();
if(Vcc>=0.95 && ie<0.0 ) control2();
if(Vcc<=.05 && ie>0.0 ) control2();
asm cli;
conv_da(); /* Performs the D/A conversion subroutine */
asm sti;
if (xx>=10) {/* Prints instant results on screen every 100 interactions */
xx=0;
gotoxy(1,1);
cprintf("VRn= %1.3f Tn= %1.4f VRi= %1.3f Ti= %1.4f ", VRn, Tn, VRi, Ti);
```

```
gotoxy(1,2);
cprintf("Time spent: %fs ", T);
gotoxy(1,3);
cprintf("canal[%3d] = % 1.3f (pu) canal[%3d] = % 1.3f (pu) ", 0, nReal, 1, iRe-
al);
gotoxy(1,4);
cprintf("nRef= %1.2f Vcontr= %1.2f iR= %1.4f ie= %1.2f ", nRef, Vcontr, iR,
ie);
gotoxy(1,5);
cprintf("Said.R.Veloc= %1.3f Said.R.Corr= %1.3f ", iRef, Vcc);
/* %1.4f V, means, real, in 1 field, with 4 decimal digits, in "V".
%3d means, integer, in 3 fields. */
}
t.stop();     /* Ends timing */
T=t.time();     /* We consider the sampling time with initial value
T=0.003 s. If the running time of "t" greater than "T" program, must attempt
to reduce it. If "t" is smaller we do T=t.*/
}
while (tecla!=27);
}
```

Acknowledgements

The authors would like to thank UNIFEI electrical engineering undergraduate students Ricardo Nogueira Brasil and Thiago Augusto de Mello Araújo, as well as designer José Vander da Silva and English teacher João Castilhos at Speaking, Itajubá/ MG, Brazil, for assistance in the preparation of this work concerning the translation of the paper from Portuguese into English.

The authors also wish to thank everyone who directly or indirectly contributed to the success of this research study.

Author details

Angelo José Junqueira Rezek[1*], Carlos Alberto Murari Pinheiro[1],
Tony Youssif Teixeira Darido[1], Valberto Ferreira da Silva[1], Otávio Henrique Salvi Vicentini[1],
Wanderson de Oliveira Assis[2] and Rafael Di Lorenzo Corrêa[2]

*Address all correspondence to: rezek@unifei.edu.br

1 UNIFEI – Federal University of Itajubá, Itajubá–MG, Brazil

2 Mauá School of Engineering, Mauá Institute of Technology, Brazil

References

[1] Knospe, C., "PID Control", IEEE Control Systems, v. 26, n.1, p. 30-41, 2006.

[2] Ming-fu, Z., Yan, C., Zhi-yuan, Z., Chun, D., Yu, D., "Research of Complex Fuzzy Control on-off Magnetism Team Motor Speed-Adjusting System", 5th Conference in Power Electronics and Motion Control Conference, 2006, IPEMC/CES/IEEE '06.

[3] Buxbaum, A., Schierau, K., Straughen, A. "Design of control systems for DC drives", Spring-Verlag Berlin Heidelberg, p.162-177, 1990.

[4] Fröhr, F., Orttenburger, F., "Introduction to the Electronic Control" (Spanish), Siemens, Marcombo S.A., Barcelona, España, 1986.

[5] Pinheiro, C. A. M., Gomide, F. A. C., "Frequency response design of fuzzy controllers", VII IFSA World Congress, v. 3, p. 434-439, 1997.

[6] Pinheiro, C. A. M., Gomide, F. A. C., "On the equivalence between basic fuzzy and classical controllers", VIII IFSA World Congress, v. 2, p. 594-597, 1999.

[7] Pinheiro, C. A. M., Gomide, F. A. C., "Fuzzy control of nonlinear systems by learning method and frequency response", FUZZ-IEEE Conference, v. 1, p. 444-448, 1999.

[8] Pinheiro, C.A.M., "Analysis and Project of Fuzzy Control Systems: An Approach at the Frequency Domain", Campinas, Unicamp, (PhD Thesis, Portuguese), 2000.

[9] Zadeh, L. A., "Fuzzy Sets", Information and Control, v. 8, p. 338-353, 1965.

[10] Rezek, A. J. J., Pinheiro, Carlos Alberto Murari, Darido, T. Y. T.,Silva, Valberto Ferreira da, Vicentini, O. H. S., Assis, W. O., "Comparative performance analysis for digital regulators in series DC motor controlled drive'', International Journal of Power and Energy Systems, v. 30, p. 15-22, 2010.

[11] Thadiappan Krishnan,T., Bellamkonda Ramaswani,B., "A Fast-Response DC Motor Speed Control System", IEEE Transactions on Industry Applications, v. IA-10 Nº 5 pp. 643-651, September/October, 1974. Control System", IEEE Transactions on Industry Applications, v. IA-10 Nº 5 pp.643-651, September/October, 1974.

[12] Rezek, A. J. J.; Rodrigues, M. S.; Miranda,V. A. M., Oliveira, V. A.; Cassula, A. M., Costa Jr., R. A.,Torres, A. Z., "Design and simulation of a controlled DC drive system" (In Portuguese) - II SIMEAR, II International Seminar of Electrical Motors and Drives, EPUSP, São Paulo - SP – Brazil, proceedings II SIMEAR, p. 141-160, 1991.

[13] Gustavo G. Moraes; Daniel C. G. Ferreira; Geraldo S. F. Júnior, "Speed control of a DC machine" (Portuguese), undergraduate course final end of course work (electrical engineering) - Federal University of Itajubá, supervisor : prof. Dr. Angelo José Junqueira Rezek, 2013.

[14] Tony Youssif Teixeira Darido,T.Y.T., "Digital regulators performance comparative analysis in series excited DC motor" (In Portuguese), Electrical engineering MSc dis-

sertation – Federal University of Itajubá (UNIFEI), supervisor: Angelo José Junqueira Rezek, 2004.

[15] Silva,D.S; Gonçalves F., "Digital implementation of fuzzy controllers" (In Portuguese), undergraduate course, end of course work (electrical engineering) - Federal University of Itajubá, supervisor: prof. Dr. Carlos Alberto Murari Pinheiro, December 1999.

IntechOpen