Rev. Téc. Ing. Univ. Zulia. Vol. 34, Nº 1, 77 - 85, 2011

On real-time velocity control of DC motors by using computer-aided control system design

Marlen Meza-Sánchez[1]***, Javier Moreno-Valenzuela***[2]

[1]*CICESE, Carretera Tijuana-Ensenada No. 3918, Zona Playitas, Ensenada, B.C., 22860, Mexico. marmeza@cicese.mx.*

[2]*Instituto Politécnico Nacional-CITEDI, Ave. del Parque 1310, Mesa de Otay, Tijuana, B.C., 22510, Mexico. Fax: +52 664 623 1388. moreno@citedi.mx*

Abstract

In this paper, the advantages of a computer-aided control system design tool, specifically Matlab, are addressed to achieve real-time velocity control of direct current (DC) motors. To this aim, an experimental set-up is proposed, which avoids the use of a data acquisition board by employing the parallel port and two microcontrollers to establish the data feedback between the computer and the DC motor. Attention to the concepts of fast computing and real-time computing is also paid. Experiments confirmed the advantage of using a computer-aided control system design tool as well as the differences in the performance of a control algorithm implemented using fast computing and real-time computing.

Key words: DC motor, velocity control, Matlab, fast computing, real-time computing.

Sobre control de velocidad en tiempo real de motores de CD usando diseño de sistemas de control asistido por computadora

Resumen

En este artículo, las ventajas de una herramienta de diseño de sistemas de control asistido por computadora, específicamente Matlab, es discutida con el fin de lograr control de velocidad en tiempo real de un motor de corriente directa (CD). Para lograr este fin, se propone una plataforma experimental la cual evita de uso de una tarjeta de adquisición de datos al emplear el puerto paralelo y dos microcontroladores para establecer la retroalimentación de datos entre la computadora y el motor de CD. Se presta especial atención a los conceptos de computación rápida y computación en tiempo real. Los experimentos confirmaron las ventajas de usar una herramienta para el diseño de sistemas de control así como las diferencias entre las prestaciones de un algoritmo de control implementado usando computación rápida y computación en tiempo real.

Palabras clave: motor de CD, control de velocidad, Matlab, computación rápida, computación en tiempo real.

Introduction

A control designer must fulfill the following steps in order to design a suitable control system for a plant [1]: 1) design a controller, 2) analyze and modify it if the specifications are not satisfied, 3) choose hardware and software and implement the controller, 4) test and validate the control system, and 5) tune the controller on line, if necessary. To accomplish these steps, the use of interactive multi-windowed computer-aids in instruction (interactive modules) is particularly significant because it provides practical insight into control system fundamentals. In essence, an interactive module in real-time control systems is a collection of graphical windows whose components are active, dynamic and clickable. By using a high-speed real-time computing platform it is possible to implement many of the control algorithms reported in papers that only consider theoretical aspects of modeling and control. Moreover, developments in automated code generation allow users to create real-time code from graphical, control system simulation software (e.g., Matlab/Simulink). These tools enable researchers to focus on control system design, implementation, and evaluation rather than on time-consuming, low-level programming. In addition, a variety of educational/research plants are now commercially available from different vendors that capture the multidisciplinary nature of the field (e.g., robot manipulator, inverted pendulum, magnetic levitation, water tank, pH control rig, helicopter, ball and beam, direct current (DC) motor). However, despite the recent interest in incorporating technological advancements into control systems laboratory courses, a control laboratory experience is not commonplace [2].

Frequently, many computer science and electrical engineering students have the misconception about real-time computing is equivalent to fast computing. The objective of fast computing is to minimize the average response of a given set of tasks. However, the objective of real-time computing is to meet the individual timing requirement of each task. Rather than being fast, the most important property of a real-time system should be predictability; that is, its fundamental and timing behavior should be as deterministic as necessary to satisfy the system specifications. Fast computing is helpful in meeting stringent timing specifications, but fast computing does not guarantee predictability [3]. Therefore, if a controller is implemented believing that fast computing can guarantee the achievement of the control objective then the result can be completely different to the one expected.

In this paper, the problem of real time velocity control of DC motors is revisited. However, novel ingredients are incorporated with a didactic point of view. In first instance, state-of-the-art PC technologies are combined to develop a system-specific interactive real-time control system, which are useful for enhancing both research and education. Specifically, this paper describes the velocity control of a DC motor by using a computer-aided control system design tool consisting in PC / WindowsXP / Matlab. Let us notice that Matlab incorporates the Simulink and Real-Time Workshop toolboxes, which permit to the control system engineers and students to analyze, design, and visualize the performance of controllers via real-time. Through a user-friendly graphical interface, users can modify the controller and signal parameters on-the-fly contrary to typical implementations where modifications are code-based [4, 5]. Besides, one major feature of the Matlab environment is the ability to access a number of toolboxes (e.g., optimization, system identification, signal processing, etc.). The proposed set-up does not require a data acquisition board. Instead, data feedback is accomplished via parallel port and microcontrollers, which is important from the didactic point of view, since computer science and electrical engineering students can practice number of microcontroller routines and exploit the capability of the parallel port to be used as a data acquisition system.

In addition, the problem of the misconception about fast computing and real-time computing is also revisited. Let us notice that Simulink is enabled with the so-called normal mode of computing, which is based in a time vector that is not connected to a hardware clock, then the calculations are done as fast as the computer can, which is interpreted as a fast computing mode that depends on the physical characteristics of the PC. In addition, Simulink is also equipped with the external mode of computing, which is available via Real-Time Workshop toolbox, which allows the real-time execution of the constructed code [6]. In other words, a way in that the misconception about fast computing and real-time computing can be understood under the light of velocity control experiments in a DC motor is provided, addressing fundamental issues that are of concern to all control system designers.

PI velocity control of DC motors

A classical linear description of a direct current (DC) motor considering the voltage as the input is given by [7-9],

$$J\ddot{q} + F_v\dot{q} = kv, \tag{1}$$

where $\dot{q}(t)$ denotes the motor shaft velocity, $J > 0$ is the motor shaft inertia, $f_v > 0$ is the viscous friction coefficient, $k > 0$ is the so-called motor constant, and v is the voltage control input.

The velocity control problem consists in designed a voltage control input v so that the velocity error $\dot{e}(t) = \dot{q}_d - \dot{q}(t)$, where $\dot{q}_d$ is a constant that specifies the desired velocity and $\dot{q}(t)$ is the motor shaft velocity, achieves the limit

$$\lim_{t\to\infty} e(t) = \lim_{t\to\infty}\left[\dot{q}_d - \dot{q}(t)\right] = 0. \tag{2}$$

A proportional-integral velocity controller is defined by [5, 6]

$$v(t) = k_p\dot{e}(t) + k_i\int_0^t \dot{e}(t)dt, \tag{3}$$

where k_i and k_p are constants that denote the integral and proportional gains of the controller, respectively. In Figure 1, a block diagram of a DC motor model (1) in closed-loop with the classical PI velocity controller (3) is shown. By using the Routh-Hurwitz criterion [7, 8], it is possible to show that the velocity error $\dot{e}(t)$ satisfies the limit (2) by using $k_i > 0$ and $k_p > 0$.

An experimental set-up

Existing work in this area have reported designs of real-time control systems, as in [2, 10, 11]; nevertheless, the need for the use of data acquisition (DAQ) boards for feedback between the controller and the system to be controlled arises. A variety of PC-based DAQ boards are available from vendors as National instruments [12], Quanser [13], Advantech [14], Data Translation [15], among others. And it does not end here, in order to provide programming environments to support the implementation of measurement and control algorithms, several vendors also provide support for DAQ boards solutions. Despite the advantages of this model, many PC-based DAQ boards tend to be expensive. Our experimental set-up considers data acquisition via parallel port. Contrary to DAQ solutions, the use of PC ports to implement data feedback is inexpensive and enables the use of microcontrollers for signal manipulation. Furthermore, total cost of

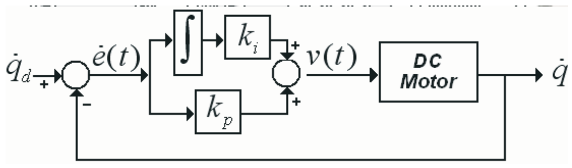

Figure 1. Block diagram of a DC motor in closed-loop with a PI velocity controller.

the proposed scheme is at least half the price of a DAQ-based solution; i.e., a couple of hundreds compared to $500 for a cheap DAQ board. Let us remark that the use of DAQ boards for feedback is independent of real-time execution and clock-based software/hardware is necessary. Moreover, DAQ-based implementations can perform non-real-time applications [16].

The experimental set-up, shown in Figure 2, has the following main components: An armature-controlled DC motor, a 14207S008 LOCG Pittman DC servo motor [17], two Basic Stamp 2 (BS2) microcontrollers from Parallax [18], an H-bridge HB25 Motor Controller from Parallax [19], and a dual positive-edge triggered D flip-flop [20].

The BS2 microcontrollers are used to interpret and apply the controller output to the DC motor. One microcontroller allows receiving data from the parallel port corresponding to the controller's output and generating the corresponding pulse for the H-Bridge. The H-Bridge is in charge of regulating voltage supplied for the DC motor. The D Flip-flop receives the signals for the motor encoder and determines the shaft direction and it is connected to the second microcontroller. This microcontroller is in charge also of reading the encoder output of the DC motor and determines the speed of the DC motor. DAQ boards function in typical designs is emulated by the combination of parallel port and BS2 microcontrollers in the proposed set-up; the parallel port allows communication and BS2 microcontrollers acquire/generate and process signals in order to enforce appropriate feedback.

Let us observe that each microcontroller is dedicated to one only task; either to receive data and convert to PWM or to read encoder signal and measure the number of counts. This characteristic avoids the need to program a scheduler or kernel in order to guaranteed hardware real-time ex-

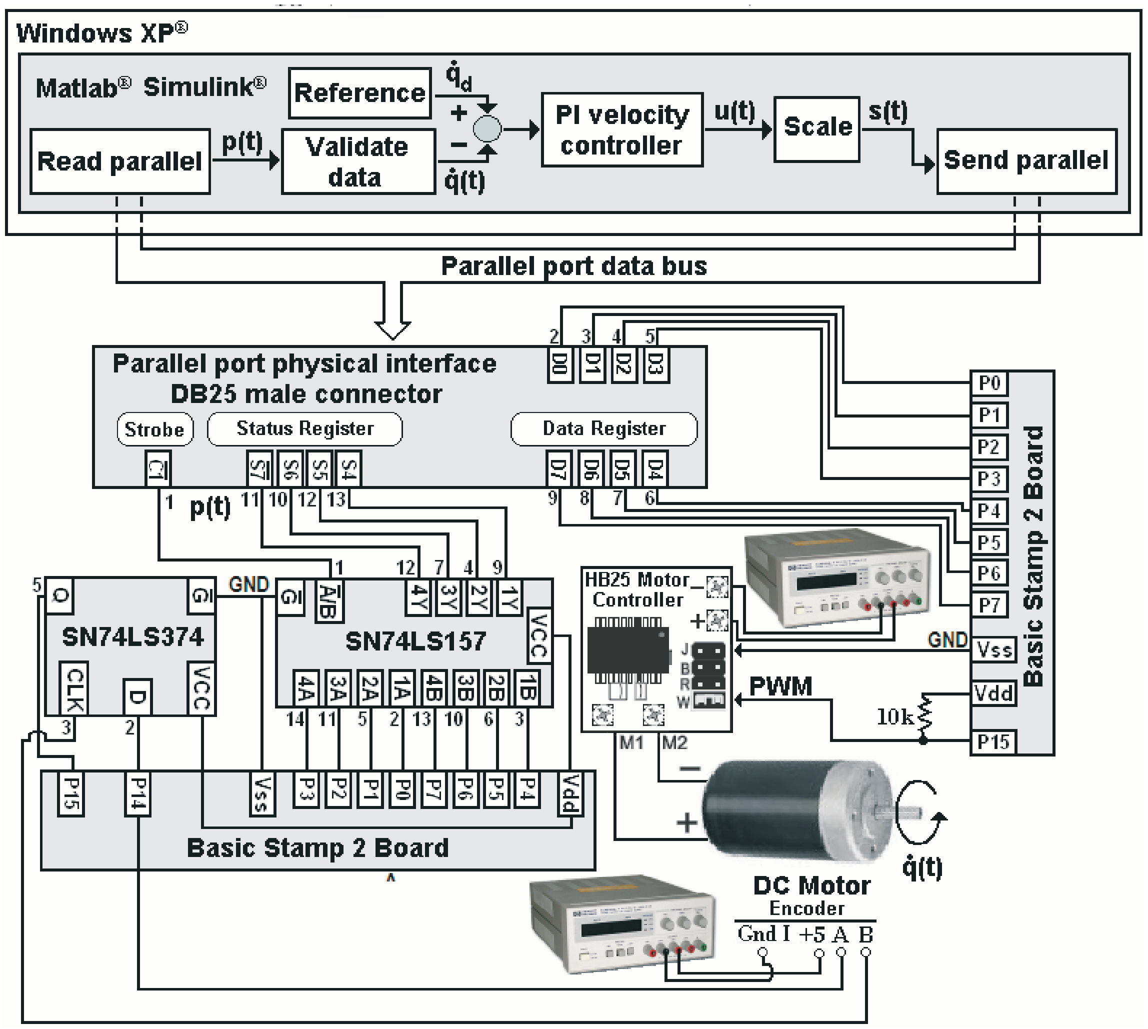

Figure 2 Block diagram of the proposed experimental set-up.

ecution as in typical microcontrollers set-ups [21, 22]. Each BS2 microcontroller performs its task in the necessary rate in order to fulfill real-time execution requirements of the software. BS2 microcontrollers use clock-based functions in order to guarantee timing and HB25 controller operates with frequency of 0.01[s] which is the minimum supported; i.e., lower sampling-time is supported by BS2 microcontrollers but it is necessary to replace the HB25 controller.

By using the proposed hardware, the main idea is that the PI velocity controller (3) should be programmed in a PC equipped with Windows XP/Matlab/Simulink/Real-Time Workshop, as illustrated in Figure 2. Thus, a proper Simulink stateflow, able to get communicated with the parallel port, should be designed.

Simulink executing modes

Simulink can execute the stateflow model in two modes, which are mostly used: normal and external. The calculations in normal mode are based in a time vector that is not connected to a hardware clock, then the calculations are done as fast as the computer can. On the other hand, the external mode concerns applications in real-time and, depending on the specified sampling time, interruptions are generated so that the calculations of the program model can be done in the specified time.

The parallel port data feedback should be incorporated in the Simulink stateflow model in different ways for each execution model. Figure 3 shows the Simulink stateflow of the implementation of the PI velocity controller with fast computing (normal mode) while Figure 4 describes the stateflow of the PI velocity controller with real-time computing (external mode). In both stateflows, Validate data and Scale blocks remarks the Simulink blocks corresponding to the necessary process in order to adequate received/send data through parallel port. In the case of normal mode, the Matlab Function blocks correspond to m-file functions which allow parallel port communication; for external mode, rec_motor and motor_send correspond to C functions embedded into S-Function Builder blocks. Let us notice that S-functions allow incorporating used-defined blocks in the Simulink stateflow model and can be converted along with the stateflow model in real-time application blocks, contrary to Matlab's built-in application functions, which can only be used in the normal execution mode [6]. In the proposed design, S-function Builder block for external mode allows including C programming language for parallel port communication; by double-clicking in it and providing the header and C files that will be used in each block, the embedded function can be generated. Furthermore, Matlab Function blocks for normal mode only need to specify the corresponding m-file function. The described Matlab functions for normal mode, C functions for external mode and Basic Stamp 2 programs are provided in Appendix A.

Results

For convenience, we have decided to measure the motor shaft velocity in units of [pulses/10ms]. Then, 1 [pulses/10ms] = 100 [pulses/s] = 1.26 [rad/s], since the encoder has a resolution of 500 [pulses/rev]. In the experiments the PI velocity controller (3) was imple-

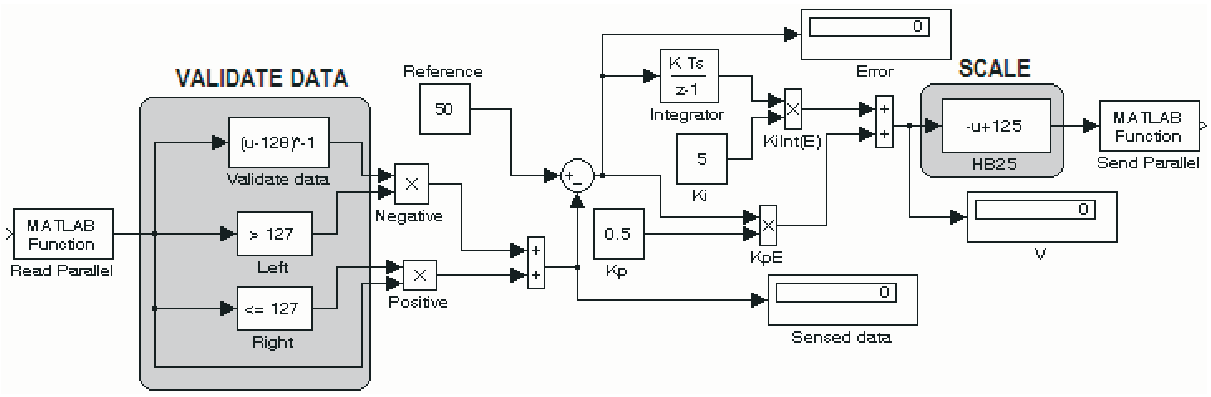

Figure 3. Simulink stateflow for the normal mode implementation of the PI velocity control, which includes the Matlab function blocks for the communication with the parallel port.

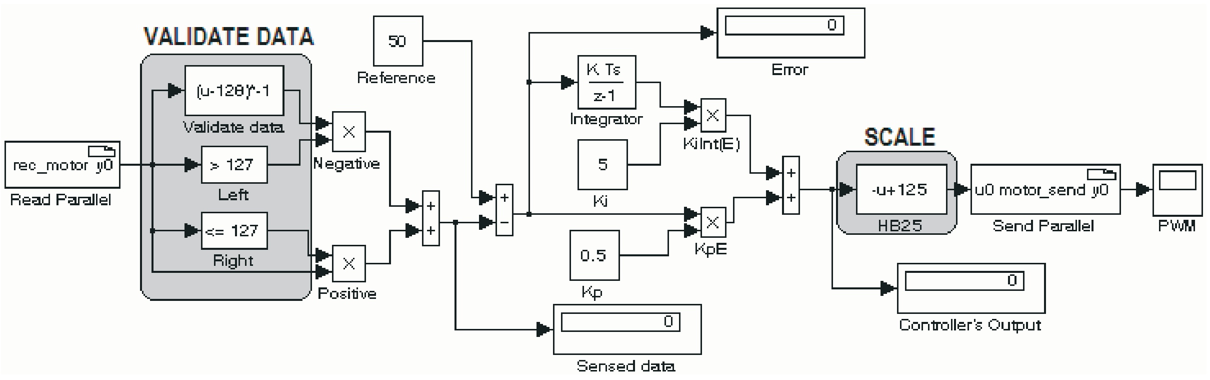

Figure 4. Simulink stateflow for the external mode implementation of the PI velocity control, which includes S-functions for the real-time communication with the parallel port.

mented with gains $k_p = 0.5$ and $k_i = 5.0$. Gains values were selected accordingly to observed experimental behavior for smooth system response. The desired velocity was

$$\dot{q}_d = 50 \text{ [pulses/10ms]} = 63 \text{ [rad/s]}. \tag{4}$$

The first set of experiments consisted in implementing the normal mode Simulink stateflow in Figure 3. Three runs of the model were done using the sampling times t_s = 0.01, 0.03, and 0.05 [s]. Secondly, we implemented the PI velocity controller in external mode Simulink stateflow in Figure 4 using the same sampling times. The results are presented in Figure 5 and 6. In Figure 5, it is easy to observe that some "jumps" in the time response are presented in the implementation of the normal mode, which are non typical in linear systems. On the other hand, in despite of the different sampling times, the different implementations in the external mode presented a behavior of the time response consistent with linear system theory, i.e., exponential convergence of the actual $\dot{q}(t)$ velocity to the desired one, $\dot{q}_d$, see Figure 6.

Discussions

The fast computing implementation of the PI velocity controller (3) through the normal mode Simulink stateflow shown in Figure 3 showed unsatisfactory results. The reason of the ``jumps'' in the time responses illustrated in Figure 3 when implementing the normal mode is that the calculations are done without using any physical clock and no especial priority to control process is given when the computer is executing the model.

With the external mode Simulink stateflow shown in Figure 4, we were able to implement the PI velocity controller (3) using real-time computing. The experiments presented excellent repeatability, contrary to the case of the fast computing implementation. The reason is that the constructed stateflow model is compiled and converted into an executable file to be run in real-time by the operating system kernel.

In the normal mode setting (fast computing), the Matlab functions for the parallel port cannot be used in real-time execution due inter-

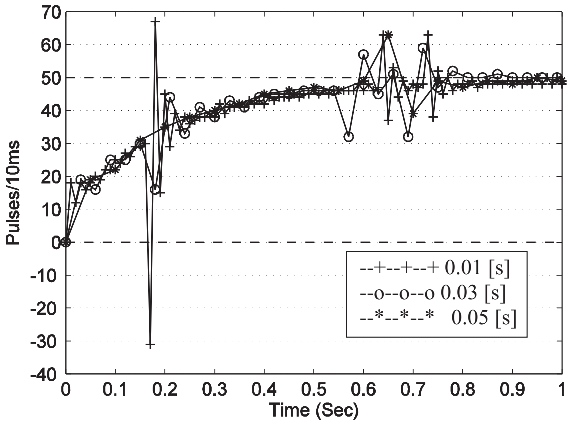

Figure 5. Normal execution mode: Experimental results using different sampling times.

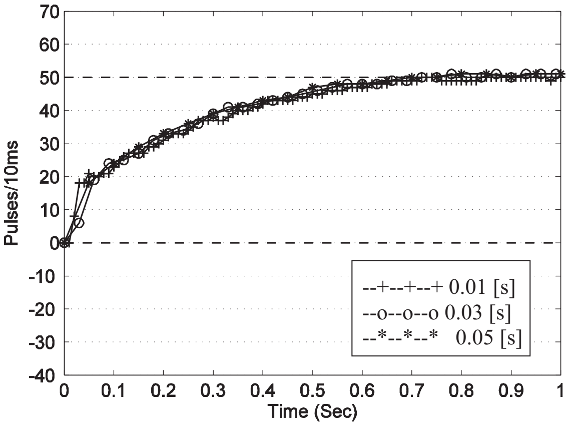

Figure 6. External execution mode: Experimental results using different sampling times.

action with Windows's Scheduler which operates accordingly to hierarchical execution policies [23]. Moreover, the fast computing implementation increases the duration of the experiment when increasing sampling-time in the model. For example, the following approximated results were obtained by timing a 5[sec] normal mode execution for the PI velocity controller using a Windows XP Pro SP2 Intel Pentium IV 3.00 GHz with 512Mb of RAM: a) sampling-time of 0.05[s] gives a 77.5[s] of duration, b) sampling-time of 0.03[s] results in 155.1[s], and finally, c) sampling-time of 0.01[s] gives a measured 467.3[s] of duration. On the other hand, external mode (real-time execution) the duration of execution is guaranteed regardless sampling-time provided PC characteristics can handle the operation weight; otherwise, model will not be loaded and executed by Simulink in this mode.

Thus, we have shown that implementing a control algorithm with a fast computing focus can lead to a result not necessarily useful in practical applications. Furthermore, the importance of guaranteeing the timing requirement of each task was confirmed.

Summary

In this paper, a control scheme in real-time without using a data acquisition board has been proposed. The proposed scheme is based on using the PC parallel port and two microcontrollers to achieve data feedback and compared to common microcontrollers developments provides several advantages: a) no need to learn languages for low-level programming, b) no clock-based schedulers experience or knowledge for design is necessary, c) any control algorithm can be implemented providing correct use of toolboxes, d) no need to download data to microcontrollers when changing control algorithm, and e) friendly block-based interface for complex control algorithms design. By implementing a PI velocity regulator in a DC motor, comparison between fast and real-time computing have been carried out. The results showed that real-time computing is crucial in the implementation of control algorithms, while the fast computing can be used for simple monitoring of data. Due simplicity and control algorithm independence, we expect that the proposed experimental set-up can be useful in control systems education and research.

Acknowledgements

This work was partially supported by SNI-CONACyT, and SIP-IPN.

References

1. Sanchez, J., Dormido S. and Esquembre F., “The learning of control concepts using interactive tools”, Computer Applications in Engineering Education, Vol. 13 (2005) 84-98.
2. Dixon W.E., Dawson D.M., Costic B.T., and de Queiroz M.S., “A Matlab-based control system laboratory experience for undergraduate students: toward standardization and shared resources”, IEEE Trans. on Education, Vol. 45 (2002) 218-226.
3. Stankovic J.A., “Misconceptions about real-time computing: A Serious Problem for Next-Generation Systems”, Journal Computer, Vol. 21 (1988) 10-19.
4. Y. Tipsuwan and M. Chow, “Fuzzy logic microcontroller implementation for DC motor speed control”, Proc. of 25th annual conference of the IEEE Industrial Electronics Society IECON’99, Vol. 3 (1999) 1271-1276.
5. Tang J., “PID controller using the TMS320C31 DSK with online parameter adjustment for real-time DC motor speed and position control”, Proc. of the IEEE International Symposium on Industrial Electronics, Vol. 2 (2001) 786-791.
6. The MathWorks, Inc., “Real-Time Workshop© User’s Guide”, available at: http://www.mathworks.com/help/pdf_doc/rtw/rtw_ug.pdf accessed November, 2010.
7. Kuo B.C.: “Automatic Control Systems”, 4th Ed., Prentice Hall, Englewood Cliffs, 1982.
8. Dorf R.D. and Bishop R.H.: “Modern Control Systems”, 8th Ed., Addison-Wesley, Menlo Park, 1998.
9. Spong M.W. and Vidyasagar M.: “Robot Dynamics and Control”, John Wiley, New York, 1989.
10. Monroy C., Kelly R., Arteaga M. and Bugarin E., “Remote Visual Servoing of a Robot Manipulator via Internet2”, Journal of Intelligent and Robotic Systems, Vol. 49 (2007) 171-187.
11. Yao Z., Costescu N.P., Nagarkatti S.P., and Dawson D.M., “Real-Time Linux Target: A MATLAB-Based Graphical Control Environment”, Proc. of the 2000 IEEE International Symposium on Computer- Aided Control System Design, (2000) 173-178.
12. National Instruments Corporation, Official Website. http://www.ni.com/ accessed November, 2010.

13. Quanser Inc., Official Website. http://www.quanser.com/ accessed November, 2010.

14. Advantech Corporation, Official Website. http://www.advantech.com/ accessed November, 2010.

15. Data Translation, Inc., Official Website. http://www.datx.com/ accessed November, 2010.

16. Kim B.K., "Control engineering education with experiments on real-time control system implementation", Proc. of the 17th world congress of the International Federation of Automatic Control, Vol. 17, Part 1 (2008) 9105-9110.

17. ClickAutomation, "PITTMAN LOCG 14000 series", available at: http://www. click automation.com/products/index.php?func=show&pid=361&cid=144 accessed November, 2010.

18. Parallax, Inc., "Basic Stamp 2 Module", available at: http://www.parallax.com/ detail.asp?product_id=BS2-IC accessed November, 2010.

19. Parallax, Inc., "HB25 Motor Controller", available at: http://www.parallax.com/ dl/docs/prod/motors/HB-25MotorController-V1.2.pdf accessed November, 2010.

20. University of Colorado at Boulder. "DM74LS74A- dual positive-edge-triggered D flip-flops with preset, clear and complementary outputs", available at: http://ece-www.colorado.edu/ ~mcclurel/dm74ls74a.pdf accessed April, 2010.

21. Lara-Rojo F., Sánchez E.N. and Zaldívar-Navarro D., "Minimal fuzzy microcontroller implementation for didactic applications", Journal of Applied Research and Technology, Vol. 1, No. 2 (2003) 137-147.

22. Marau R., Leite P., Velasco M., Martí P., Almeida L., Pedreiras P. and Fuertes J. M., "Performing flexible control on low-cost microcontrollers using a minimal real-time kernel", IEEE Trans. on Industrial Informatics, Vol. 4, No. 2 (2008) 125-133.

23. Christensen P., "The truth about Windows real-time architectures", Website, 2007. Available at: http://www.ardence.com/assets/9b4bbe2c6b644403bdff09f60c415e16.pdf accessed November, 2010.

Recibido el 13 de Abril de 2010
En forma revisada el 17 de Enero de 2011

Appendix A. Experimental platform source code

pport fcn.c: Function developed in C for sending and receiving data through parallel port in external mode.

```
#include "conio.h"
double read_motor ()
{
unsigned int in_val;
outp(0x37a, inp(0x37a) | 0x01);
in_val = ((inp(0x379)^0x80) > 4);
outp(0x37a, inp(0x37a) & 0xFE);
in_val = in_val | ((inp(0x379)^0x80)>4) < 4 ;
return (double) in_val;
}
double send_motor(double dato)
{
if (dato < 0)
{dato = 0;}
if (dato > 255)
{dato = 255;}
outp(0x378,(unsigned int) dato);
return dato;
}
```

pport_fcn.h: Header file for pport_fcn.c

```
double send_motor(double dato);
double read_motor ();
```

PBasic program for the Basic Stamp microcontroller which allows sending data to parallel port. It determines speed and direction of the CD motor.

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' {$PORT COM4}
cha PIN 14
dirlect VAR Word
cycles VAR Word
```

```
DIRL = %11111111
DIRH = %10000000
Main:
COUNT cha,10, cycles
IF (cycles > 127) THEN cycles = 127
IF (dirlect = 1) THEN DIR = 128 ELSE dir = 0
IF (cycles = 0) THEN dir = 0
OUTL = cycles + dir
GOTO Main END
```

PBasic program for the Basic Stamp microcontroller for receiving data through the parallel port. It provides the length of the pulse to be generated by the HB25 microcontroller.

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' {$PORT COM4}
HB25 PIN 15
pulsos VAR Byte
Main1:
PULSOUT HB25,750
GOTO Main
END
Main:
pulsos = INL
PULSOUT HB25, pulsos * 2 + 500
PAUSE 8
GOTO Main
```

read_par.m: Matlab function for receiving data through parallel port in the normal mode execution.

```
function y = read_par(t)
DIO = digitalio('parallel','LPT1');
Status=addline(DIO,1:4,1,'in');
Control.Strobe=addline(DIO,0,2,'out');
putvalue(Control.Strobe,1)
tmp1 = xor(getvalue(Status),[0 0 0 1]);
tmp1 = [tmp1(1) tmp1(2) tmp1(3) tmp1(4) 0 0 0 0];
putvalue(Control.Strobe,0);
tmp2 = xor(getvalue(Status),[0 0 0 1]);
tmp2 = [0 0 0 0 tmp2(1) tmp2(2) tmp2(3) tmp2(4)];
binary = tmp2 + tmp1;
y = binvec2dec(binary);
```

write_par.m: Matlab function for sending data through parallel port in the normal mode execution.

```
function write_par(x)
parport=digitalio('parallel','LPT1');
out_lines=addline(parport,0:7,0,'Out');
if x < 0
x = 0
end if x > 255
x = 255
end
putvalue(out_lines,x);
```