Make a PI controller on an 8-bit micro

Crescencio Hernandez-Rosales, Ricardo Femat-Flores, and Griselda Quiroz-Compean

1/4/2006 1:06 PM EST

This article shows you how to implement a classical PI (proportional-integral) controller on a simple 8-bit microcontroller. To implement the PI controller, we developed specific libraries that make it possible for the microcontroller to perform arithmetic operations with 16- and 32-bit precision. Such resolution is necessary to reduce the steady-state error of the system being controlled. One advantage of this configuration is that it can be programmed into microcontrollers with less than 128 bytes of RAM and 4KB of ROM on chip. This design has been used to control a direct current (DC) gear motor but can be used to control other kind of actuators as well. Experimental results show a good performance of the overall embedded system.

Why would you use a PI (proportional-integral) controller instead of PID (proportional integral-derivative) controller? The PI controller is commonly used when the reference signal given to the system are steps (set-points). On the other hand if the reference signals imposed to the system are ramps or other kinds of time-functions, it's better to use a PID controller; nevertheless, in practice the derivative term could amplify disturbances input or noise as the PID is not well tuned. This can prompt oscillations or the system can become unstable.

Why use an 8-bitter?

Currently, several manufacturers make 16- and 32-bit microcontrollers (MCUs) with features that enable easy control of almost any process of medium complexity. Eight-bit microcontrollers still dominate the market, however, because of their small size, low cost, and simple programming. Because of these advantages, 8-bit MCUs are found in process control, automotive, industrial, and appliance applications, among many others.[1, 2] Some of the newer MCUs provide clock speeds from 4 to 40MHz; 64KB of internal flash memory and 1KB of RAM in some models; on-chip analog-to-digital converters (ADCs), digital-to-analog converters (DACs), or pulse-width modulator (PWM) outputs; a watchdog timer; 16-bits timers; and serial or USB ports.

A few examples of these MCUs include:

• The enhanced 8051 from Intel, series 87C51RA/RB/RC

• The W78E858 by Winbond Electronics, which is compatible with the Intel 8052

• The P89V51RD2 chip from Phillips Semiconductors; its main features are the 64KB of flash and 1KB of RAM on chip; in addition to its PCA block, the P89V51RD2 is composed of four modules that can be configured as high speed I/O ports, compare/capture registers, PWMs, or watchdog timers [3]

• The DS5000T series from Dallas Semiconductors; its main advantage is the 32KB of internal SRAM that can be partitioned as the user likes into data or program memory; in addition, it has a real-time clock on chip making it suitable for data logging applications [4]

• The enhanced flash USB microcontroller series PIC18FX455/ X550 from Microchip incorporates 32KB of internal flash memory, ADCs, EAUSART, and USB V2.0 interfaces, making it, in our opinion, perfect for connectivity applications [5]

Although the features of 8-bit MCUs are continually improving, in most cases these new features are ignored by designers because they're using the chips for the control of states, which don't require the newer features.[1, 2] Recently a novel method has been used to exploit these kinds of MCUs by using them in a closed-loop configuration aided with the well-known classical control theory. Examples of work on this topic are available in the literature.[6, 7, 8] In such applications the authors demonstrated that feedback control improves the control of some DC motors. It's important to mention that, in the examples, except in Johnston,[7] the realization of the PID (proportional-integral-derivative) controllers were implemented using 16-bit MCUs in Hitex's paper [6] or, as in Neary,[8] where an integrated data acquisition system particularly the model ADuC845 by Analog Devices, was used.

Until few years ago, these kind of tasks (micro-positioning or servo control) had been addressed using digital signal processors (DSPs), mainly because such devices are faster and have higher precision than the 8-bits MCUs. However, some applications don't require high precision or the team simply can't justify the cost of a DSP. It's usually cheaper to use an 8- or 16-bit MCU without diminishing performance.

For both commercial and education reasons, we developed an alternative to control a DC gear motor in a closed-loop configuration using a standard Atmel AT89C52 device. This chip is a general-purpose 8-bit MCU without some of the features we mentioned earlier.[3, 4, 5] However, in this application, we show that a discrete-time control can be implemented into this simple MCU. Nevertheless a little of knowledge about the closed-loop is needed to apply this alternative. The following section describes some concepts of the control theory. Details can be found in the endnotes of this article.[9, 10, 11, 12]

Feedback-control theory

Figure 1 shows the block diagram of a closed-loop system. In this configuration a portion of the information is fed back from the process and subtracted from the reference signal in order to calculate the error signal. This error signal is used by the PID to adjust the control input such that the process output can reach the given reference.

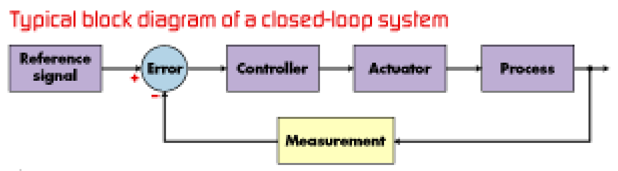

Figure 1

View the full-size image

In this diagram, the *process block* refers to the physical system to be controlled. The *controller block* is an electronic device (a microcontroller) that calculates the necessary energy to modify the process such that the control objective can be reached. Usually, in a control scheme the *controller* precedes an actuator (for instance, a motor, turbine, valve, resistance, and so forth) so that the process can be manipulated. Finally, to close the loop, a *measurement block* is needed, which is often a sensor or transducer that provides a signal to automatically compare the actual state of the system variables with the desired objective.

Some of the motivation for using this configuration was to diminish the effects of parameter variations coming into the system, reduce the effects of disturbance inputs, improve the time transient responses of the process output, and to compensate for the steady-state error.[9] Another advantage of this approach is that the effects of the "dead zone" in the motors can be reduced.

The control problem

Our goal is to control the angular velocity of a DC gear motor by using a classical discrete-time PI-like controller. Figure 2 provides a block diagram of the sample data system. In the figure, $r(kT)$ denotes the reference signal, $u(t)$ is the control voltage applied to the motor, $x(t)$ stands for the system output, $y(kT)$ represents the discrete signal generated by the sensor (angular displacement), $V(kT)$ is the output signal of the Velocity Sensor (VS), $e(kT)$ is the error defined by the difference $r(kT)–V(kT)$, $d(t)$ and $n(t)$ are, respectively, the disturbance input to the plant and the noise in the sensor.[10] Here (kT) is used to represent a discrete signal with a sample-time given by T.

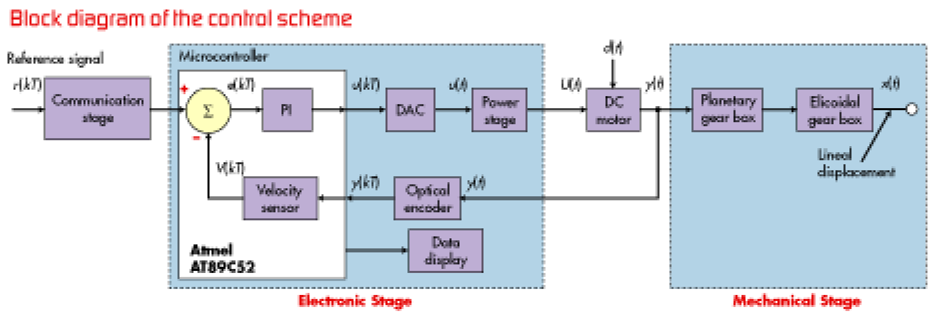

Figure 2

View the full-size image

The control algorithm works like this. First, a reference signal is given to the MCU via a serial port. At this point the MCU enables the VS to count the pulses per revolution provided by the motor's optical encoder. A few milliseconds later, the MCU computes the $V(kT)$, which is directly proportional to the motor's velocity. Using this signal, the MCU then calculates the error signal $e(kT)$ and the voltage $u(kT)$ by means of the PI-like controller, which is the control input in discrete-time. Such a signal is then converted into analog form, $u(t)$, by means of the DAC.[13] The control input $u(t)$ is current-amplified by the power stage and applied to the motor. This loop is repeated as many times as necessary until that the error signal $e(kT)$ is near to zero. As the control objective is reached, the loop is terminated and the MCU waits for a new reference.

Figure 3a shows the schematic diagram of the sample-data system from Figure 2. This design incorporates an LCD module where it's possible to display some data, such as the motor velocity. Figure 3b shows the printed circuit boards designed to control the DC gear motor. The board on the left contains the AT89C52-based control unit, while the board on the right holds the power supplies. Together these boards provide the output stage to directly control the motor.

a) Schematic diagram of the control system, b) Electronic boards designed for the embedded system

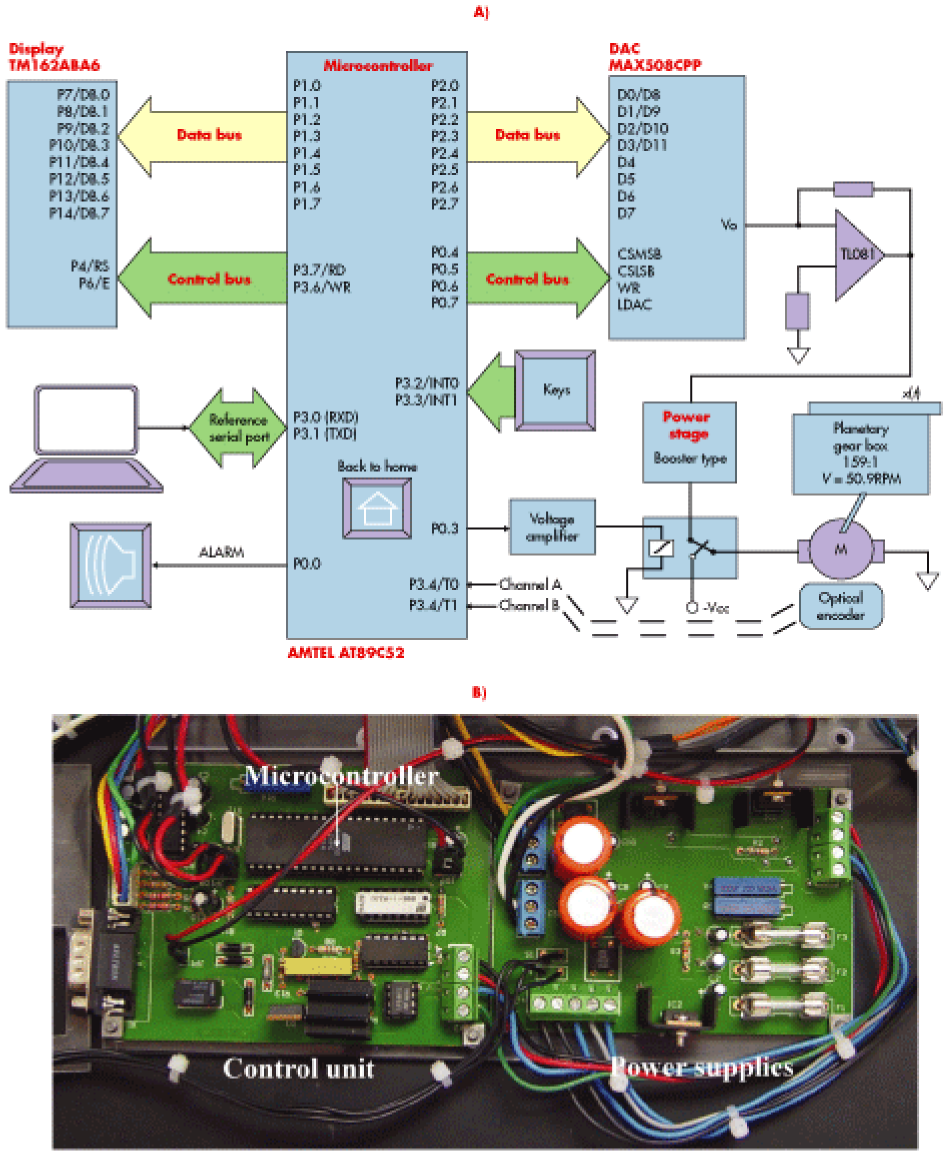

Figure 3

[View the full-size image]

Microcontroller-based control stage
This block contains the control unit and represents the brain of the electronic stage because it controls the other blocks. The AT89C52 includes 8KB of flash memory and 256 bytes of RAM, 32 I/O pins, three 16-bit counter/timers, and a UART. Programmed into the on-chip memory are the control algorithm (PI) and the velocity sensor routines.

Digital-to-analog stage
In this stage a Maxim MAX508 chip performs the digital-to-analog conversion with 12 bits of resolution. Through on-chip resistors, the DAC can be configured for one of the three output voltage ranges, from 0V to +5V, 0V to +10V, or 0V to ±5V, with a ±15V

dual power supply. Digital data $u(kT)$ is loaded into the input registers of the DAC in a right-justified (8+4) format. This allows an easy interface with the AT89C52 MCU. The resolution voltage is given by $u(t) = V_{\mathrm{REF}}[(u(kT))/2^{11})]$, where V_{REF} is the internal voltage reference equal to +5V, $u(kT)$, the input code (calculated by the PI algorithm), and $u(t)$ the analog voltage supplied to the motor.[13] In this application the operation range is from 0 to ±10V.

Power stage
The power stage is implemented using a TIP-41C (NPN power transistor) in a "booster" current-amplifier configuration with an operation range from 0 to +10V. In this stage the DAC output current is amplified from 10mA to 6A to provide the current required by the motor. This stage also includes auxiliary circuitry to supply to the motor a negative voltage of -10.47V to return it to its home position. This circuitry is implemented by using a PNP power transistor TIP-42.

Plant (gearmotor)
The motor in this application is a 12W DC gear motor by Faulhaber MicroMo, model 2342S012CR, with a nominal voltage of 12V. The motor provides 637 oz/in of torque at normal load with a gear-reduction ratio 159:1. Speed at load is 5.3RPM at 12V, with maximum current consumption at load of 75mA. The control input is a DC voltage and the feedback output is the angular velocity $V(t)$ in RPMs.[14]

Optical encoder measurement stage
In this stage the motor velocity is measured by means of the VS and an optical encoder (a HEDS5540G from Hewlett-Packard) that provides 360 pulses per revolution (PPR) in channels A and B, and 1 PPR in a third channel called Index. The VS (programmed into MCU) works as follows: the A channel of the optical encoder is connected to the external counter/timer T0 of the MCU and timer 2 is programmed with a capture time of 2ms, during which counter/timer 0 is incremented for each digital pulse provided for the encoder. Thus VS is measured in pulses per millisecond.

Figure 4 shows a comparison between the theoretical and actual motor velocity. The theoretical velocity comes from the manufacturer's data sheet while the actual velocity was measured using VS. Note on this graph there is an offset between these two signals. This difference is attributable to the nominal parameters of the motor. To corroborate the precision of VS, we measured the motor velocity using a high-precision oscilloscope and obtained results that were similar to VS.

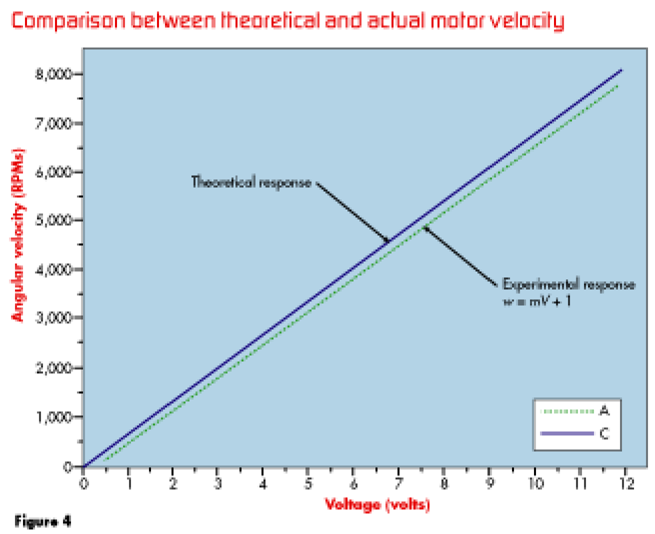

Figure 4

View the full-size image

Communication stage (ADC)
This simple stage handles the communication between the computer and the motor-control unit by using the microcontroller's built-in serial port. The baud rate chosen was 9,600 bits per seconds, and a standard Maxim MAX232 chip converts the voltage level of the UART to RS-232 levels.

Mechanical stage (gear box)
This block is composed of two mechanical stages, a planetary gear box assembled with the motor, and another external gear box formed by a pair of worm gears with reduction ratios of 159:1 and 625:1. The main function of these stages is to reduce the angular velocity of the motor and to translate the angular motion to linear motion as shown in Figure 2.

Creating the Motor-Control Model

To apply well-establish classical control theory, it's necessary to designate in advance the mathematical model of the system to be controlled with the purpose of simulating its dynamic behavior in open-loop as well as closed-loop modes. Otherwise, the designer may not have enough experience to tune the parameters of the PID controller, and its closed-loop performance could be poor or, in the worst case, unstable. Depending on the mechanical system, this may cause injuries to the system or to the user. For this reason, we recommend simulating the system's behavior before planning its physical implementation.

In general terms, a *model* is a set of differential equations that represents an approximation of the physical system's dynamics. One classic method to identify the model of simple systems is to excite it with a known input and measure the response. The inputs applied and the outputs measured depend on the nature of the system and the variables that the designer needs to control in the process. In applying these control techniques, we characterized our motor by applying voltages of 2.5V, 5V, 7.5V, and 9.47V over similar intervals of time to measure the motor's response to such inputs.

Identifying the Model Input-Output (Cause-Effect)

Figure 5 shows the open-loop motor response when four steps were applied to its terminals. These signals show the angular velocity reached by the motor with such inputs and represent the steady-state gain of the system. Note that the motor response is similar to the simple-lag system response when it is excited with a step input; therefore, the motor model can be represented by Equation 1; see Chapter 4 of Ogata.[12]

a) Open-loop motor response for four steps input of 2.5V, 5V, 7.5V, and 9.47 V

b) Comparison of the nominal model response and the actual motor response

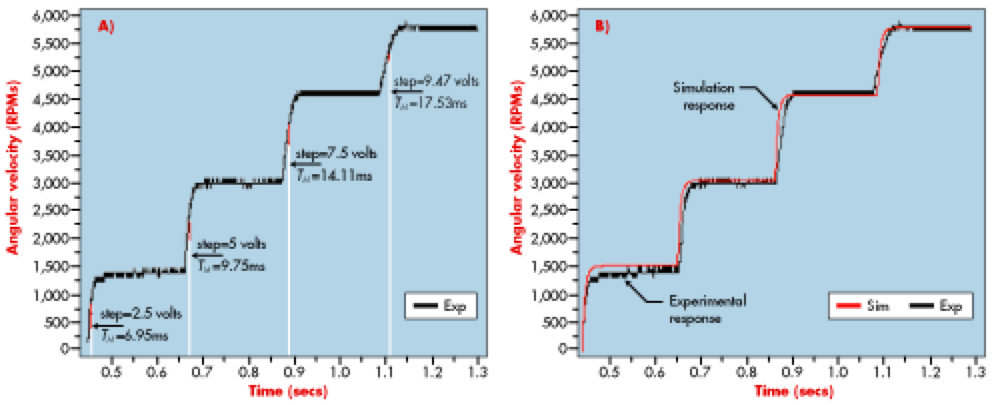

Figure 5

View the full-size image

Equation 1 represents the general transfer function of a first order (simple-lag) system in the Laplace domain. Here K_M is the *steady-state gain* of the system and T_M is known as the *time-constant* of the system and is defined as the time at which the system output reaches the 63.2 % of the steady-state value.

$$G_m = \frac{K_M}{(T_M)s+1}$$

(1)

According to the definition of T_M and K_M, these values can be obtained from the open-loop motor response in Figure 5a.

Equation 2 shows the transfer functions obtained for each step applied.

$$G_1 = \frac{566.576}{(6.95)s+1}, G_2 = \frac{660.238}{(9.75)s+1},$$
$$G_3 = \frac{611.072}{(14.11)s+1}, G_4 = \frac{607.18}{(17.53)s+1}$$

(2)

where the values of T_M and K_M substitued for G_1 through G_4 are in milliseconds and RPMs respectively.

However, note that T_M and the steady-state gain K_M are a little bit different for each step applied. For this reason, and without losing generality, we can take the arithmetical average of these parameters to obtain one nominal model that can approach as near as possible the real motor dynamic. Equation 3 displays the model obtained with this approximation. For more details on the identification of this model, see our report.[15] Figure 5b shows a comparison between the nominal model response simulated (red line) and the actual motor response (black line). From this graph we can deduce that the nominal model is capable of reproducing the motor dynamic and can therefore be safely used to tune the PI parameters.

It's important to point out that the inputs given to the motor model in simulations were similar in magnitude and time to the voltage given to the motor in the experimental identification.

$$G_M = \frac{611.26}{(12.085)s+1}$$

(3)

Once we've derived the nominal model of the motor (Equation 3), we proceed to tune the parameters of the PI controller.

Tuning the PI Controller Parameters

Currently, many stability-analysis techniques are available, including the well-known Laplace method, frequency-response, Nyquist criterion, Bode representation, root-locus method, and others.[9, 10, 12] In this example, the selection of the PI parameters is based on root-locus design using the control system toolbox of MatLab. These parameters are chosen such that the closed-loop system is stable and so the system response can be fast without any overshoot.

Figure 6 shows the closed-loop configuration of the system in the Laplace domain and includes the motor and controller-transfer function. The control law used is a classical PI, where T_i is known as the *restoration time* of the integral action, K_p the *proportional gain*, and K_p/T_i the integral gain.

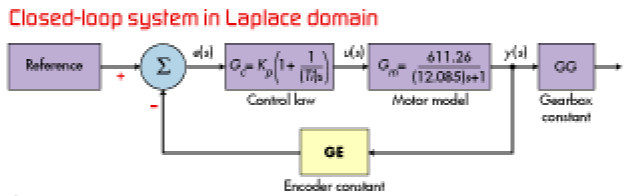
Closed-loop system in Laplace domain
Reference
+
-
Σ
e(s)
$G_c = K_p\left(1+\frac{1}{T_i s}\right)$
Control law
u(s)
$G_m = \frac{611.26}{(12.085)s+1}$
Motor model
y(s)
GG
Gearbox constant
GE
Encoder constant

Figure 6

View the full-size image

Figure 7a shows the simulated closed-loop motor response for different values of K_p and using the model given by Equation 3, with a T_i = 50ms. In this graph, it's easy see that, for values greater than 1, the system response is faster than with values of $K_p < 1$. However, for $K_p > 1$ the motor response exhibits undesirable overshoot that can reduce the useful life of the motor. In this example, the gain selected was $K_p = 0.5$, although $K_p = 0.7$ can be selected too. Figure 7b shows the simulated control input computes by the PI controller, such that the motor can reach the reference specified (6,000 RPM). Note that these signals are less than or equal to 10V, due to the saturation block included to simulate the DAC's output range. The following section shows the experimental motor response with the gains selected.

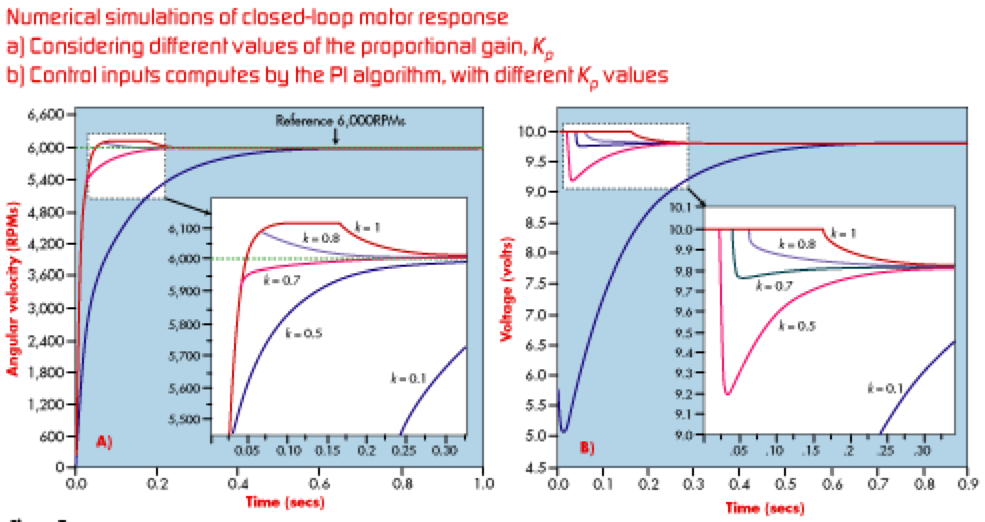

Figure 7

View the full-size image

Implementation of the PI Controller

PID controllers are widely used in the process-control industry, mainly because of their effectiveness and simple structure.[16]

Equation 4 shows the expression of a PID controller in continuous time, where $u(t)$ is called the *control signal* and $e(t)$ is the *error signal*. K_p, K_i, and K_d are the proportional, integral and differential gains, respectively.

$$u(t)=K_p e(t)+K_i\int_0^t e(t)dt+K_d\frac{d}{dt}e(t)$$

(4)

In this example a PI controller is used, with $K_d = 0$ and $K_i = K_p/T_i$. Taking the derivative of Equation 4, we obtain:

$$\dot{u}(t)=K_p\dot{e}(t)+K_i e(t)$$

(5)

where:

$$\dot{u}(t)=\frac{du}{dt}$$

and:

$$\dot{e}(t)=\frac{de}{dt}$$

.

However, this expression is difficult to implement in a microcontroller, so we can approximate the derivative by mean of Equation 6, known as the finite-difference approach, where ΔT is the sample time.

$$\dot{u}(t)\cong\frac{u_{k+1}-u_k}{\Delta T};\dot{e}(t)\cong\frac{e_{k+1}-e_k}{\Delta T}$$

(6)

Substituting Equation 6 in Equation 5, we obtain the discrete-time PI controller.

$$u_{k+1}=u_k+K_p\left(e_{k+1}-e_k\right)+K_i\Delta T\left(e_k\right)$$

(7)

Once reducing Equation 7 and substituting the PI parameters selected, K_p = 0.5, T_i = 50ms, with a sample time ΔT = 10ms, the discrete-time representation of the PI controller is obtained and can be implemented in the microcontroller.

$$u_{k+1} = u_k - 0.4e_k + 0.5e_{k+1}$$

(8)

It's important to point out that in this application, the tuning of PI parameters was made in continuous time. However, the stability in discrete time must be analyzed with the parameters chosen too, mainly because the PI is implemented in a digital processor. In this analysis we found that the closed-loop poles are located in the unit circle (according to Routh's stability criterion for discrete-time systems), which means that the closed-loop system shown on Figure 6 is stable. In other words, this means the system is feasibly safe. For more details of this test, read Quiroz-Compen's report.[17]

Figure 8a shows the experimental motor response using Equation 8 in closed-loop configuration. The reference given to the MCU was 6,000 RPM. In practice, the motor reaches the reference in 0.01 min (600ms) and without overshoot. Nevertheless, the close-up in Figure 7a shows that there is a steady-state error, although its magnitude represents 1.3% of the steady-state value, which is acceptable in practice. Figure 8b shows the control input computes by the PI and applied to the motor for reach the reference imposed.

Motor closed-loop control
a) Experimental motor response for a reference of 6,000RPMs
b) Control voltage calculates by the PI controllers for reach the reference prearranged

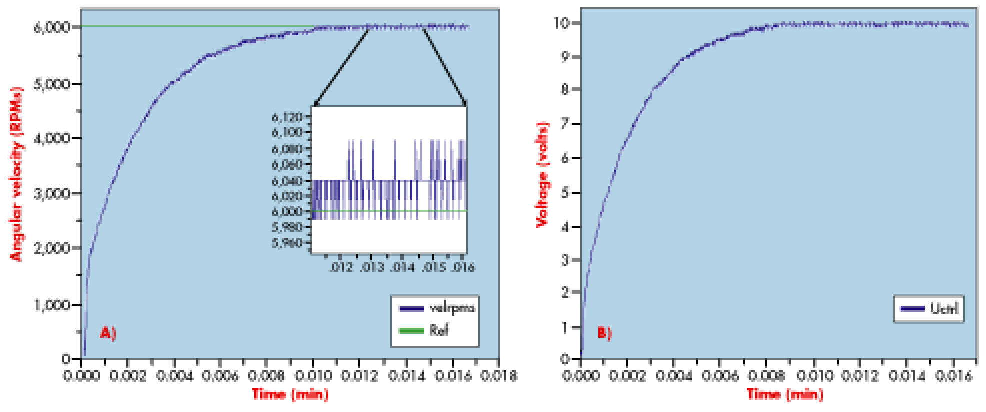

Figure 8

View the full-size image

Figure 9 shows the flux diagram of the main program implemented into microcontroller. The assembly code is not shown to save space in this article, but you can find the code online at www.embedded.com/code.

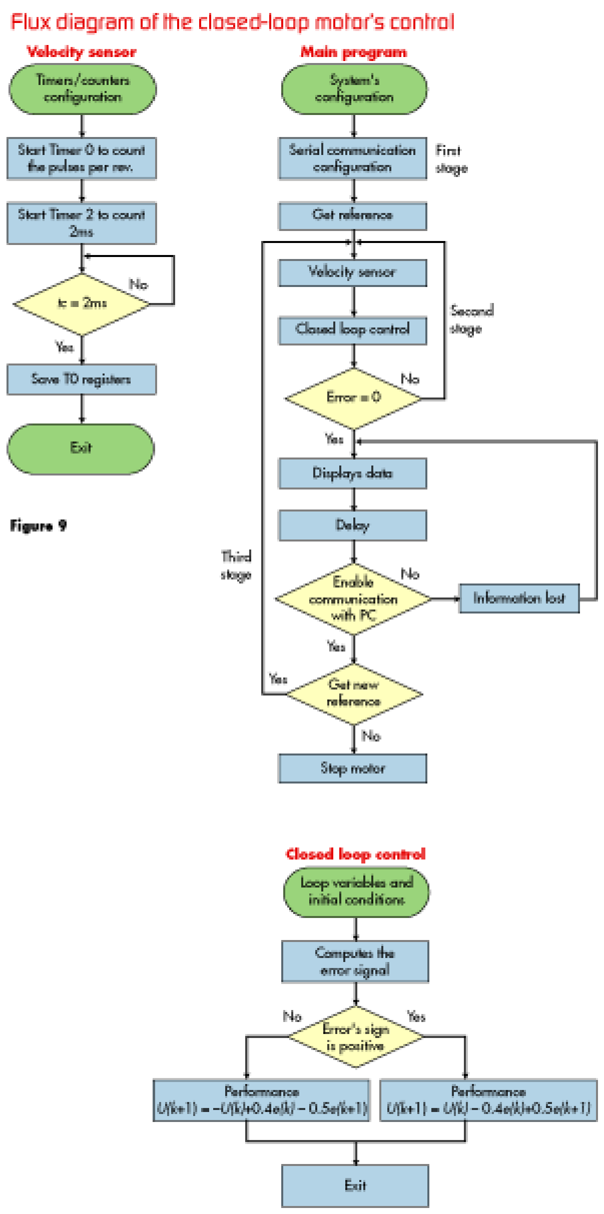
Flux diagram of the closed-loop motor's control
Velocity sensor
Timers/counters configuration
Start Timer 0 to count the pulses per rev.
Start Timer 2 to count 2ms
tc = 2ms
No
Yes
Save T0 registers
Exit
Main program
System's configuration
Serial communication configuration
First stage
Get reference
Velocity sensor
Closed loop control
Second stage
Error = 0
No
Yes
Displays data
Delay
Third stage
Enable communication with PC
No
Information lost
Yes
Get new reference
Yes
No
Stop motor
Closed loop control
Loop variables and initial conditions
Computes the error signal
No
Error's sign is positive
Yes
Performance U(k+1) = −U(k)+0.4e(k) − 0.5e(k+1)
Performance U(k+1) = U(k) − 0.4e(k)+0.5e(k+1)
Exit

Figure 9

[View the full-size image]()

In this article we showed that a feedback-control scheme can be implemented in a simple 8-bit MCU. Moreover, we've shown the use of a classic method to identify a DC motor. Despite the fact that the MCU is among the simpler ones available, the time it requires to compute the whole program and bring the motors to their maximum reference speed (9,100RPM) was only 866ms in the worst case. So we believe that using the newer MCUs in a closed-loop configuration can improve the control of some electromechanical actuators such as valves or motors and improve the performance of some processes with slow dynamic behavior.

Crescencio Hernndez-Rosales *is a laboratory technician from Institute of Research of San Luis Potosi and currently is working on the design and instrumentation of an electromechanical pump to delivery drugs to patients with Type I diabetes. He is interested in embedded systems design and the design and control of mechatronic devices. He has a degree in electrical engineering from the University of San Luis Potosi, Mexico. He can be reached at heros@titan.ipicyt.edu.mx.*

Ricardo Femat-Flores *is a professor and active researcher at the Institute of Research of San Luis Potosi. He has published 42 technical papers in international journals and five in international magazines. His current scientific interests are control and chaos theory and the regulation of the glucose concentration in diabetic patients. He can be reached at rfemat@ipicyt.edu.mx.*

Griselda Quiroz-Compen *has received the Bc. Sc. degree from Technological Institute of San Luis Potosi and the Ms.Sc. degree from Institute of Research of San Luis Potosi, Mexico in 2003 and 2005 respectively. She is currently working toward the Sciences Doctor Degree in control and dynamical systems. Her research interests include control theory applied to biomedical sciences. She can be reached at gquiroz@ipicyt.edu.mx.*

Endnotes:

1. Maurice, B. "ST62 microcontrollers drive home appliance motor technology, AN885/1196," Application Note, ST Microelectronics, 1998, www.st.com.

2. Katausky, J., I. Horder, and L. Smith. "Analog/Digital Processing with Microcontrollers," AR-526 Applications Engineers, Intel Corporation, www.intel.com.

3. Data sheet. "W78E858 8-bit microcontroller," Winbond Electronics, Rev. A4, May 2004.

4. Data sheet. "DS5000T Soft microcontroller Module," Dallas Semiconductors, www.maxim-ic.com.

5. Data sheet. "PIC18F2455/2550/4455/4550, High-Performance, Enhanced Flash USB Microcontrollers with Nano Watt Technology," Microchip Technology Inc., 2004.

6. Hitex. "Basic DC Motor Speed Control With The Infineon C167 Family." Hitex: UK. www.hitex.co.uk/c166/pidex.html.

7. Johnston, K., S. Narum, G. Bergeson, and S. Bowden. "PID motor control with the Z8PE003," application note AN003002-0401, Zilog, Inc. 2001: www.Zilog.com.

8. Neary, E. "Mixed-signal control circuits use microcontroller for flexibility in implementing PID algorithms," Analog Dialogue 38-01, January 2004. www.analog.com/analogdialogue.

9. de Vegte, J. V. Feedback control system, third ed., Prentice-Hall, New Jersey, 1994.

10. Franklin, G. F. and J. D. Powell. Digital Control of Dynamic Systems, second ed. Addison-Wesley, 1990, USA.

11. Dorf, R. C. and R. H. Bishop. *Modern Control Systems*, seventh ed. Addison-Wesley, 1995, USA.

12. Ogata, K. Ingenier'a de Control Moderna, 3 Ed. Prentice-Hall, 1998, México.

13. Data sheet. "MAX507/MAX0508, Voltage-Output 12-Bit DACs with Internal Reference," 19-4338: Rev A: 9/91, Maxim Integrated Products, Sept 1991.

14. Data sheet. "12 Watts DC-Micromotors Graphite commutation, series 2342," CR, MicroMo Electronics, www.micromo.com.

15. Hernndez-Rosales, C., G. Quiroz, and R. Femat. "Instrumentación de 2 prototipos de bombas electromecnicas para el suministro de medicamento," Reporte de investigación IPICYT-DMASC No. 003, San Luis Potos', México, Febrero 2005 (in Spanish).

16. Wang, Q. G. "PID tuning for improved performance," IEEE Transactions on Control Systems Technology, Vol. 7, No. 4, Jul 1999.

17. Quiroz-Compen. G. "Instrumentación de una bomba para suministro de insulina," tesis de licenciatura, Tecnológico de San Luis Potos', México, Septiembre 2003 (in Spanish).

Reader Response

Good article...

Another 8051 based micro you might consider are the 8051F0xx series from SiLabs. You could eliminate a lot of peripheral components (D/A, oscillators, and perhaps even some of the amplifier circuitry).

Although your assembly code is well written and commented nicely, I believe a good quality compiler and C would have made the project quicker and easier to maintain and adapt for various 8051 derivatives and other motor types.

Alternatives to using matlab for PI parameter selections should have been discussed--not everyone has MATLAB--and many that have it, don't have specialized toolboxes. For those that do have it, your MATLAB files would be helpful as well.

- Paul Calvert
Senior engineer
Radiance Technologies
Huntsville, AL