MOTOR CONTROL USING "OFF-THE-SHELF" HARDWARE AND SOFTWARE

George A. Perdikaris, Engineering & Computer Science
University of Wisconsin-Parkside
900 Wood Road
Kenosha, WI 53144-2000

E-mail: perdikar@uwp.edu; Phone: (262) 595-2489; Fax: (262) 595-2114

Abstract

A method is presented for controlling an industrial motor plant by a personal microcomputer (PC) using "off-the-shelf" hardware and software. The computer-controlled system prototype is designed and simulated using MATLAB and Simulink, products of MathWorks, Inc.[1] The actual control system is implemented in the laboratory using digital-to-analog converter (DAC) and encoder interface boards made by Measurement Computing Corporation. The plant consists of an industrial motor and drive made by the Bosch Rexroth Corporation. The computer-controlled system is run in real time using the Real-Time Workshop (RTW) and Real-Time Windows Target (RTWT) software, also made by MathWorks, Inc. Computer simulation results are verified experimentally.

I. Introduction

Industrial automation incorporating computers is becoming increasingly important in the production of goods and services. It is a highly sophisticated job to design and implement automation systems that operate and coordinate modern manufacturing processes. Quite often, such systems involve the application of computers to control the speed or position of the shaft of motors in real time.

The "spinning" of motors intelligently is fundamental to quality manufacturing. In the metal processing industry, for instance, the doors of home appliances such as refrigerators, washers, and dryers are formed out of steel, which is cut to size from large coils of metal and then pressed into shape by large presses controlled precisely by computers. In paper, plastics, wood and other industries, consumer products ranging from toilet paper and baby diapers to office furniture and automobiles are also made by controlling motors by computers.

This paper presents a practical and systematic method for modeling, simulation, and real-time control of an industrial motor plant by using a Pentium PC as host and target. The computer-controlled system is implemented in real time without having to write programs in languages such as C/C++ and/or other low-level programming languages like assembly. The interface (I/O) devices are commercially available and relatively inexpensive.

Using MATLAB and Simulink with RTW and RTWT creates a single-computer environment for prototyping and testing real-time systems. In this real-time development environment the PC is used first to create models using Simulink blocks. After simulating a model with Simulink in normal mode, executable code can be generated and run in real time with Simulink in external mode.

II. Computer Controlled Systems

A closed-loop digital control system is shown in the Simulink block diagram of Fig. 1. As can be seen from the diagram, digital-to-analog converter (DAC) and incremental encoder or motor pulse generator (MPG) interfaces have been assumed. Specifically, the output of the digital controller (manipulation) is converted into analog form by the DAC before it can be communicated to the analog plant. For the output, it is common to control the speed and/or position (displacement) of the

[1] MATLAB, Simulink, Real-Time Workshop , and Real-Time Windows Target are registered trademarks of MathWorks, Inc.

shaft of a motor by using a transducer that converts the shaft rotation into pulses that can be read by the computer.

A transducer that monitors motor speed every T seconds, where T is the control sampling time, is the incremental encoder or MPG. The pulse generator also acts as a differentiator, because its signal is proportional to the change in position from the previous sample. For instance, if the input to the MPG represents position in radians (rad), the units of the MPG gain are pulses/rev = pulses/(2πrad). Therefore, after differentiation, the velocity signal is (pulses/T). Specifically, a 5000 pulses/rev MPG has a gain, K_{dt}, given by

$$K_{dt} = \frac{\#\text{pulses}}{\text{rev}} = \frac{5000(pulses)}{2\pi(rad)} = 795.775 \text{ (pulses/rad)}$$

The Simulink simulation diagram of Fig. 1 includes block models for the DAC and MPG interface devices. These models account for the quantization effects due to the finite wordlength of the DAC, and for the differentiation as well as the discretization effects of the MPG. Using such graphical models, the wordlength and/or the input range of the DAC can be varied interactively and relevant system responses can be observed graphically or the respective values can be "captured" in real time and stored for further evaluation. The designer can also tune (interactively) controllers such as the proportional-integral-derivative (PID), integral with proportional-derivative-feedback-plus-feedforward (PDFF) – shown in Fig. 1, or other types of controllers [1][2].

III. The Laboratory Environment

The laboratory computer is a 450 MHz Pentium III running Windows 98. The PC is equipped with DAC (Analog Output), and Encoder (Encoder Input) boards made by Measurement Computing Corporation, formerly known as Computer Boards, Inc. The specific I/O boards used are the ISA-bus models CIO-DAS1602/12 and CIO-QUAD02, respectively. The first is a 12-bit, bipolar, ADC/DAC combination board while the second is a two-channel encoder input device used to monitor the motor speed/position feedback.

The plant to be controlled is a 6-pole, brushless, direct-current (DC) servomotor with rated torque of 38 Nm and speed of 2700 RPM. The motor is powered by a matching sinusoidally-commutated, pulse-width-modulated (PWM) servo drive with an integral current loop. Position, velocity, and commutation feedback is provided by a high-resolution (2^{21} increments per revolution) feedback device internal to the motor. Motor and drive are manufactured by the Bosch Rexroth Corporation. It should be noted that for our case the encoder resolution was set at 4x1250=5,000 (pulses/rev).

The Pentium PC is used as both a host and target. This software environment allows one to design, simulate, and test an application in both real time and non-real time. The PC is used with MATLAB and Simulink to create graphical models using Simulink blocks. After creating a model and simulating it with Simulink in the so-called "normal" mode, executable code can be generated with the Real-Time Workshop (RTW), the Real-Time Windows Target (RTWT), and Microsoft's Visual C/C++ compiler using the Simulink "external" mode [3][4]. During real-time execution, the Simulink model is used as a graphical user interface (GUI) for both signal visualization and parameter tuning.

With the RTW, you can quickly generate efficient C code for discrete-time, continuous-time, and hybrid systems. When used in conjunction with a rapid prototyping target such as the RTWT, the RTW provides a practical and attractive real-time development environment for testing and observing physical systems in various areas of application.

The RTWT uses a real-time kernel to ensure that the real-time application runs in real time. The real-time kernel runs at CPU ring zero (kernel mode) and uses the built-in PC clock as its primary source of time. The kernel intercepts the interrupt from the PC clock before the Windows operating system receives it. This blocks any calls to the Windows operating system. The kernel uses the interrupt to trigger the execution of the compiled model and give the real-time application the highest priority available.

The real-time kernel interfaces and communicates with input/output (I/O) hardware using I/O driver blocks. Analog-to-digital converter (ADC) or analog input (AI), DAC or analog output (AO), digital input, and digital output blocks call the drivers for input and output. Drivers also run at CPU ring zero. Communication between Simulink and the real-time application is through the Simulink external mode interface module. This module talks directly to the real-time kernel and is used to start the real-time application, change parameters, and retrieve scope data.

When running the model in real time, RTWT captures the sampled data from one or more input channels, uses the data as inputs to the block diagram model, processes the data, and sends it back to the outside world through an output channel on the I/O board.

RTWT provides a custom Simulink block library. The I/O driver block library contains universal drivers for supported I/O boards. These universal blocks are configured to operate with the library of supported drivers. You drag-and-drop a universal I/O driver block from the I/O library the same way as you would from a standard Simulink block library.

The real-time application uses the initial parameters available from the Simulink model at the time of the code generation. RTWT provides the necessary software that uses the real-time resources on the computer hardware. Based on the selected sample rate, RTWT uses interrupts to step the application in real time at the proper rate.

At each sample interval of the real-time application, Simulink stores contiguous data points in memory until filling a data buffer. Then, Simulink suspends data capture while the data is transferred back to MATLAB through Simulink external mode. Data transfer runs at a lower priority in the remaining CPU time after model computations are performed while waiting for another interrupt to trigger the next model update. Data transferred to Simulink is immediately plotted in a Simulink Scope block, or it can be saved to a MAT-file.

IV. Modeling the Motor Plant

When designing a digital control system, it is often practical and convenient to design the analog control system prototype first. The analog control system is then converted into its digital counterpart by adding appropriate interface devices and controller gains.

It is assumed that the motor feedback is monitored by an incremental encoder (MPG), which returns change in position (i.e., velocity) per sampling time T; it is also assumed that the controller output is communicated to the analog plant via a DAC.

An analog motor plant is often modeled by a double-integrator transfer function

$$G_p(s) = \frac{\square(s)}{M(s)} = \frac{K_m}{s^2} \qquad (1)$$

where $\theta(s)$ represents position, M(s) manipulation or controller output, and the analog plant gain K_m depends on the motor and its power supply parameters. If the DAC and MPG gains K_{da} and K_{dt} are taken into consideration, the digital position plant transfer function becomes

$$\frac{\theta(z)}{M(z)} = Z[\frac{1-e^{-sT}}{s}\frac{K_\omega}{s^2}] = \frac{(\frac{K_\omega T^2}{2})(z+1)}{(z-1)^2} \tag{2}$$

where the position plant gain K_θ is given by

$$K_\theta = (K_{da})(K_m)(K_{dt}) \tag{3}$$

K_θ can be calculated from given motor, drive, and interface parameter values or, better yet, it can be determined experimentally.

The digital velocity plant transfer function, $\omega(z)/M(z)$, can be determined by differentiating the digital position plant transfer function. That is,

$$\frac{\omega(z)}{M(z)} = \frac{(\frac{K_\omega T^2}{2})(z+1)(z-1)}{z(z-1)^2} = \frac{(\frac{K_\omega T}{2})(z+1)}{z(z-1)} \tag{4}$$

where K_ω is the digital velocity plant gain expressed by

$$K_\omega = TK_\omega \tag{5}$$

Like K_θ, K_ω can also be determined analytically from motor-related data or, preferably, it can be determined experimentally. K_ω represents the slope of the unit-step response of the motor plant, normally a ramp function.

There wasn't adequate information available for a mathematical description of the DC servomotor plant considered for this investigation. Thus, it was decided to identify an approximate model for the motor plant experimentally.

To obtain experimental information for modeling the motor plant, the real-time Simulink block diagram shown in Figure 2 was used. A 1-volt-equivalent pulse command was applied to the motor plant for 0.2 seconds – from 0.4 sec to 0.6 sec. The experimental results representing system input and output are shown plotted in Fig. 3.

From the input-output graphs of Fig. 3, it is obvious that a model for this plant can be approximated by an integrator whose gain is the slope of the output (ramp) – divided by the size/amplitude of the input (reference) step which is 205 pulses/T, for the 12-bit, ±10-volt DAC. Since values of the variables time, input, and output have also been stored in memory, one can determine the slope of the ramp by fitting the data points to a first-degree polynomial (straight line) using the MATLAB function 'polyfit'. This gives the open-loop velocity (integrator) gain as

slope = K_ω = polyfit(time(400:600), output(400:600),1)/205 = 1.122

Thus, assuming that T=0.001 sec, the corresponding open-loop position (double-integrator) gain becomes $K_{\square} = K_{\square}/T = 1122$.

V. Velocity and Position Control

The motor plant is controlled in real time according to the control algorithm of Fig. 4. An advantage of this control scheme is that it can be used to control velocity or position using velocity command and feedback. It can be easily modified, however, for position command and/or feedback. Note that the block marked "K_r-int" in Fig. 4 denotes software integration if the value of K_r=1, which is also the setting required for position control. If K_r=0, on the other hand, the software integrator is essentially bypassed, which is the setting if velocity, rather than position, is controlled.

For velocity or position control, the method chosen for tuning the digital PDFF controller uses the ITAE (integral time absolute error) filter forms described in Reference [1]. A MATLAB program for tuning either velocity (K_r=0) or position (K_r=1) controllers is shown in Table 1. The design of a 2nd-order low-pass digital filter is also included in the code - in case it's needed to smooth relatively high-frequency noise that may be present in the control system. For our case, the cutoff frequency for the filter was set at eight times the control system natural frequency, $\square_n$, which seemed to work well. Note that if a notch, rather than a low-pass, filter is desired, the low-pass filter can be made into a 2nd-order notch (band-reject) filter by simply changing the first numerator coefficient of the analog filter prototype (numf) from 0 to 1. In such a case, the notch frequency must be the noise frequency.

For velocity control and using the ITAE tuning criterion for a system bandwidth $\square_n$=100rad/sec, a plant gain $K_{\square}$=1.122, and a sampling time T = 0.001 sec, the program of Table 1 calculates the controller gains as

$$\text{KP} = \frac{\sqrt{2}\square_n}{K_{\square}} = 126.044, \text{ and KI} = \frac{\square_n^2}{K_{\square}} = 8912.7$$

The other gains are initially set at KV=0, KD=0, and KA = 0. If the feedforward gain KV is set equal to KP, the controller becomes equivalent to the conventional proportional-plus-integral or PI controller. Simulation and experimental plots for both cases are shown in Fig. 5. In addition to the real-time scopes, the relevant signals were "captured" in real time and saved into a data file for plotting.

Observe that the 150 (pulses/T) for the step size of the trapezoidal profile shown in the figures corresponds to a velocity command of 1800 RPM. That is, if the desired speed of the motor is 1800 RPM, the encoder gain is 5000 (pulses/rev), and the sampling time is T=0.001 sec, the speed command in (pulses/T) becomes

$$\frac{1800}{60}\left(\frac{rev}{\text{sec}}\right)5000\left(\frac{pulses}{rev}\right)\frac{0.001\text{sec}}{T} = 150 \text{ (pulses/T)}$$

Similarly for 1200 RPM, instead of 1800 RPM, the speed command becomes 100 (pulse/T).

For position control, the criterion chosen for tuning the digital controller is the ITAE filter form, also described in Reference [1]. Using this tuning criterion and assuming a position-loop bandwidth of $\square_n$=50 rad/sec, with T = 0.001 sec, we obtain the controller gains

$$KP = \frac{2.15\omega_n^2}{K_\omega} = 4.79, \quad KI = \frac{\omega_n^3}{K_\omega} = 111.41, \text{ and } KD = \frac{1.75\omega_n}{K_\omega} = 0.078$$

The other gains are initially set as KV=0 and KA = 0. If the feedforward gains KV and KA are set equal to KP and KD, respectively, the controller becomes equivalent to the more conventional proportional-integral-derivative or PID controller. Simulation and experimental plots for both cases are shown in Fig. 6. In addition to the real-time scopes, the relevant signals were also "captured" in real time and saved into a data file for plotting.

VI. Conclusions

A practical scheme for controlling an industrial motor plant has been presented. The overall control systems can be simulated on a Pentium microcomputer. The same microcomputer can also be used to run the real-time application using off-the-shelf I/O boards and software.

Using computer simulation, a designer can change the specifications of the interface devices, can substitute the pulse generator by an ADC converter, or test the control algorithm when external disturbances enter the control loop. The computer simulation results and the actual/experimental control system performance characteristics are in very close agreement.

The concepts presented or implied are general and can be applied to design and implement other types of real-time control systems.

Bibliography

[1] G. A. Perdikaris, Computer Controlled Systems: Theory and Applications, Kluwer Academic Publishers, 1991, reprinted in 1996.

[2] D. Y. Ohm, "A PDFF Controller for Tracking and Regulation in Motion Control," Proc of the PCIM Conference on Intelligent Motion, 1990.

[3] Real-Time Workshop User's Guide, The MathWorks, Inc., 1994-2001.

[4] Real-Time Windows Target User's Guide, The MathWorks, Inc., 2000.

Table 1. MATLAB program to calculate parameter values

```
% MATLAB Program to calculate controller (gain) values
% and digital filter (coefficient) values for Simulink model
wn = 100;                       % natural (control system) frequency
T = 0.001;                      % control sampling time
Kw = 1.122;                     % open-loop velocity-plant gain
Kth=Kw/T;                       % open-loop position-plant gain
Kda=10/(2^11-1);                % gain (scale) for +/- 10v, 12-bit DAC
Kdt=5000/2/pi;                  % MPG encoder gain (scale)
Km=Kth/Kda/Kdt;                 % (analog) motor plant gain
% determine tuning values for velocity/position control
% where Kr=0 ==> velocity, and Kr=1 ==> position control
Kr=0;
if Kr == 1
        % position-loop control - equate coefficients
        % ITAE tuning: s^3+1.75*wns^2+2.15*wn^2s+wn^3=
        %              =s^3+Kth*KDs^2+Kth*KPs+wn^3
        KP = 2.15*wn*wn/Kth;    % proportional gain
        KI = wn*wn*wn/Kth;              % integral gain
        KD = 1.75*wn/Kth;               % derivative gain
else
```

```
        % velocity-loop control - equate coefficients
        % ITAE tuning: s^2+sqrt(2)*wns+wn^2=s^2+Kw*KPs+Kw*KI
        KP = sqrt(2)*wn/Kw;             % proportional gain
        KI = wn*wn/Kw;                  % integral gain
        KD = 0;                         % derivative gain
end
KV = 0;                                 % velocity feedforward gain
KA = 0;                                 % acceleration feedforward gain
fn_sys=wn/2/pi;                         % control system bandwidth (Hz)
fn_f=8*fn_sys;                          % low-pass filter cutoff frequency
wn_f=2*pi*fn_f;                         % filter frequency in Hz
numf=[0 0 wn_f^2];                      % analog prototype filter numerator
denf=[1 sqrt(2)*wn_f wn_f^2];           % and Butterworth/ITAE denominator
sysa=tf(numf,denf);                     % filter transfer function
sysd=c2d(sysa,T,'prewarp',wn_f);        % digital filter transfer function
[numdf,dendf,T]=tfdata(sysd,'z');       % and numerator-denominator form
```

Fig. 1. Simulink simulation block diagram of PDFF control system

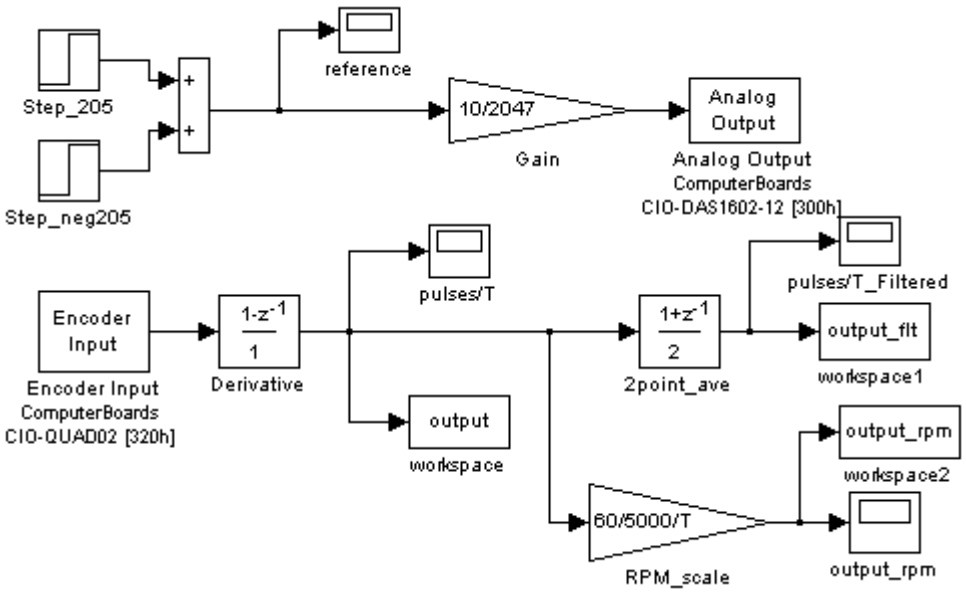
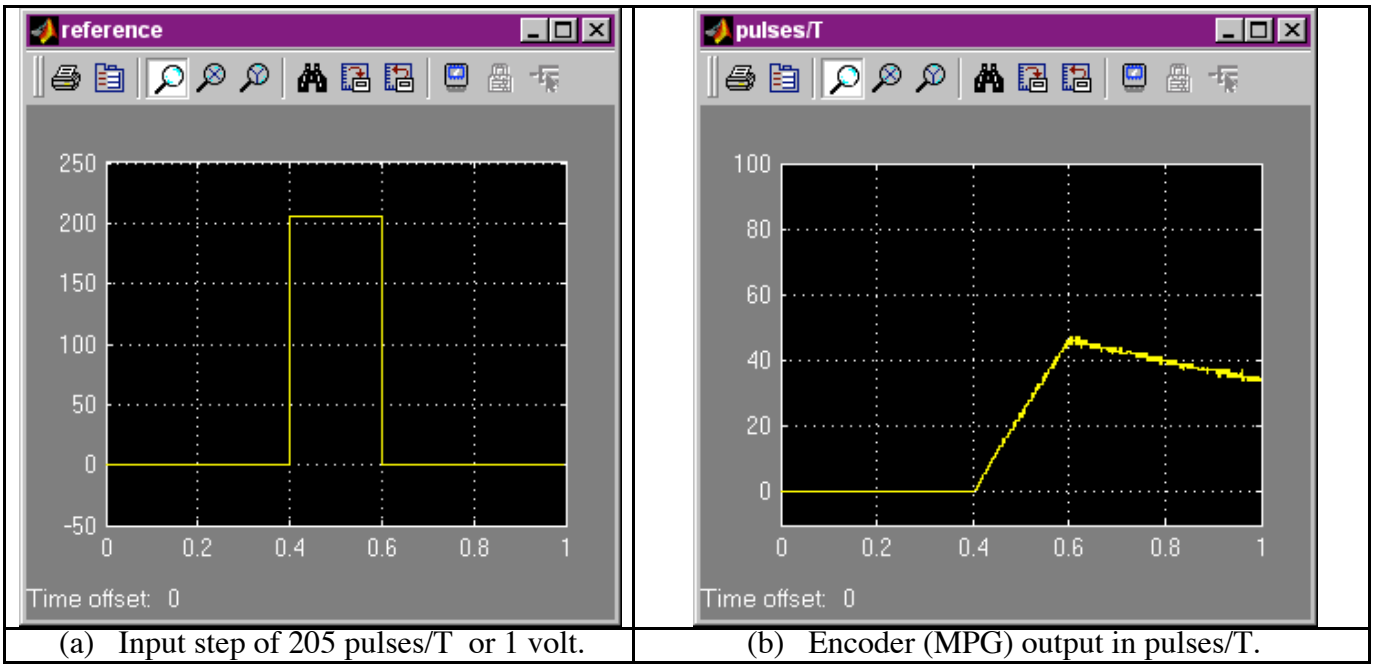


Fig. 2. Simulink (external mode) block diagram of open-loop motor plant system.



(a) Input step of 205 pulses/T or 1 volt. (b) Encoder (MPG) output in pulses/T.

Fig. 3. Experimental input-output results obtained from the real-time Simulink system of Fig. 2.

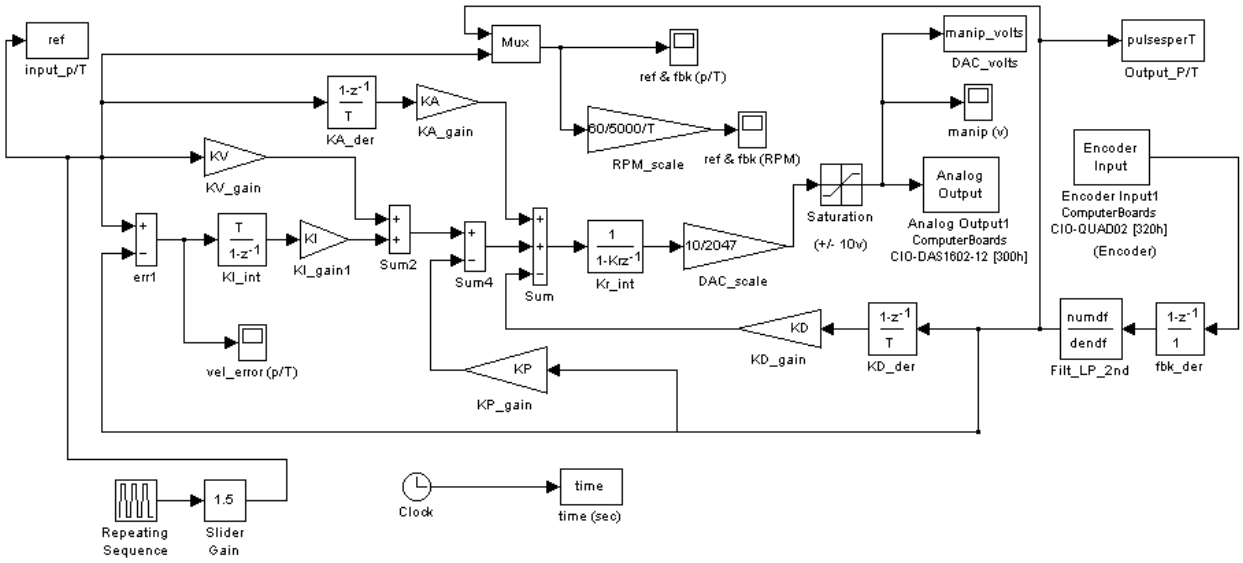
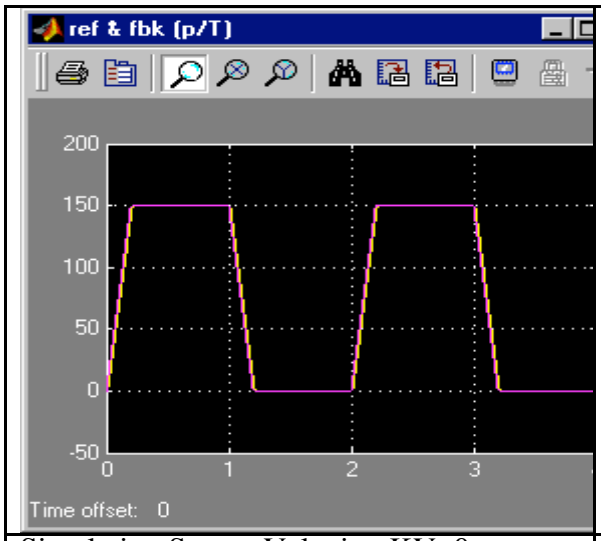
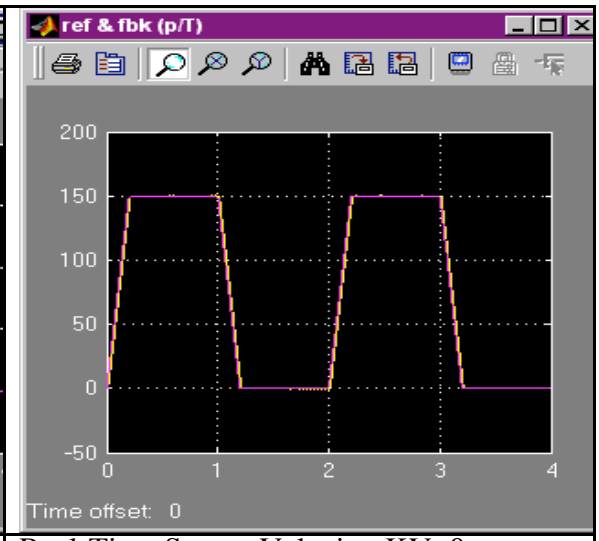


Fig. 4. Real-time Simulink block diagram of PDFF control system



Simulation Scope: Velocity, KV=0



Real-Time Scope: Velocity, KV=0

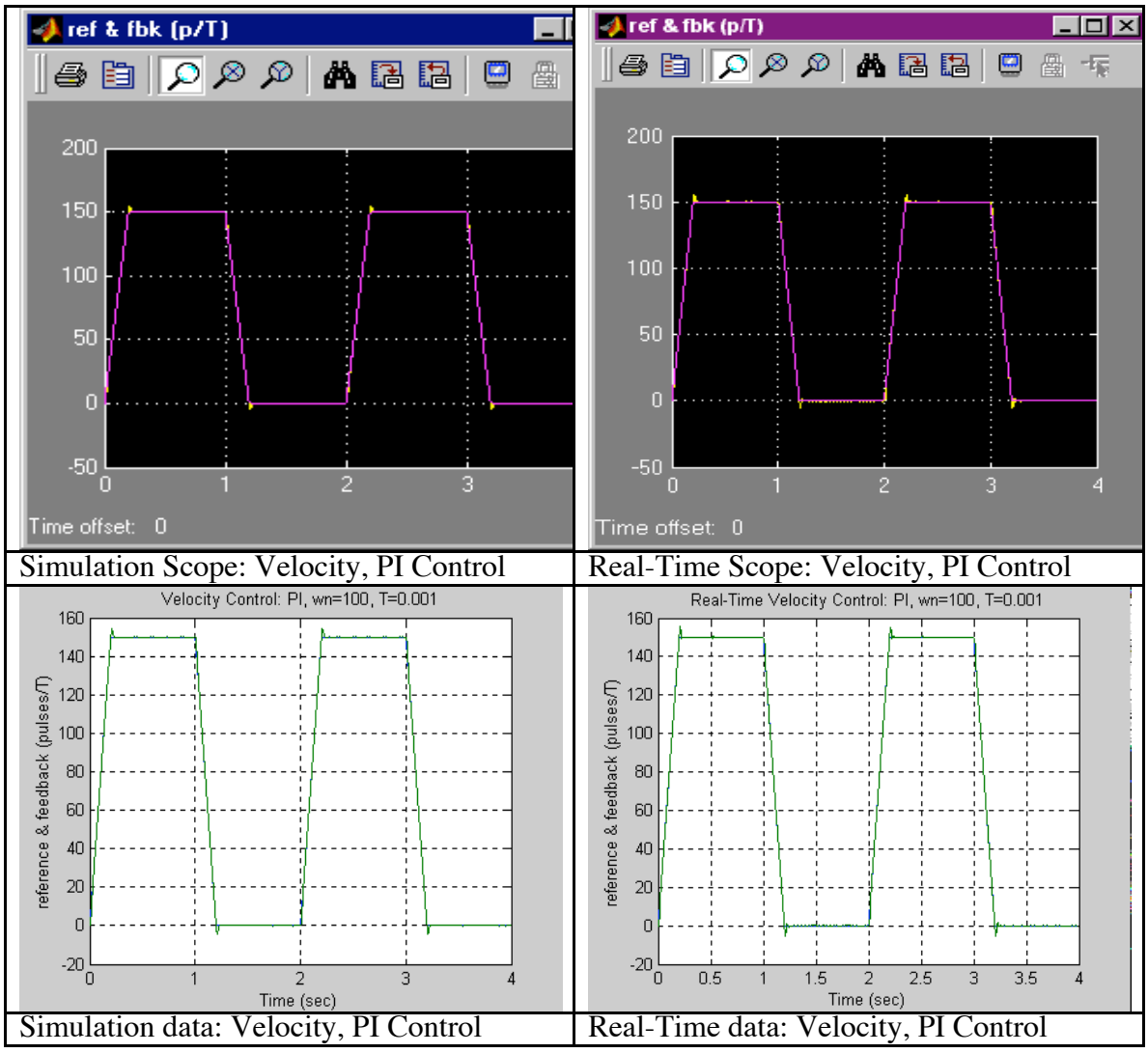
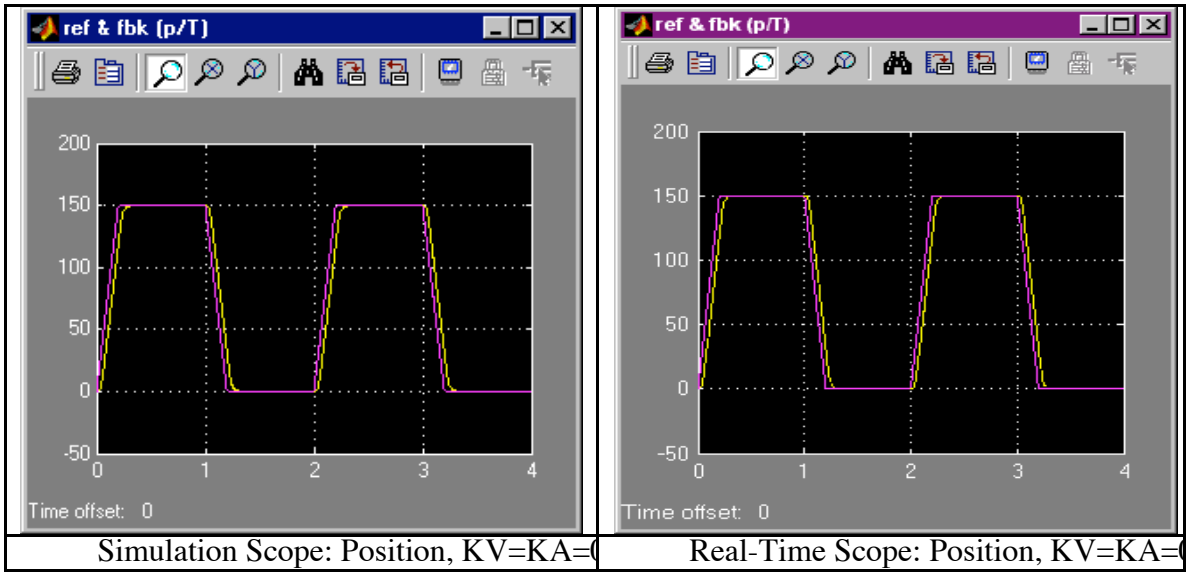
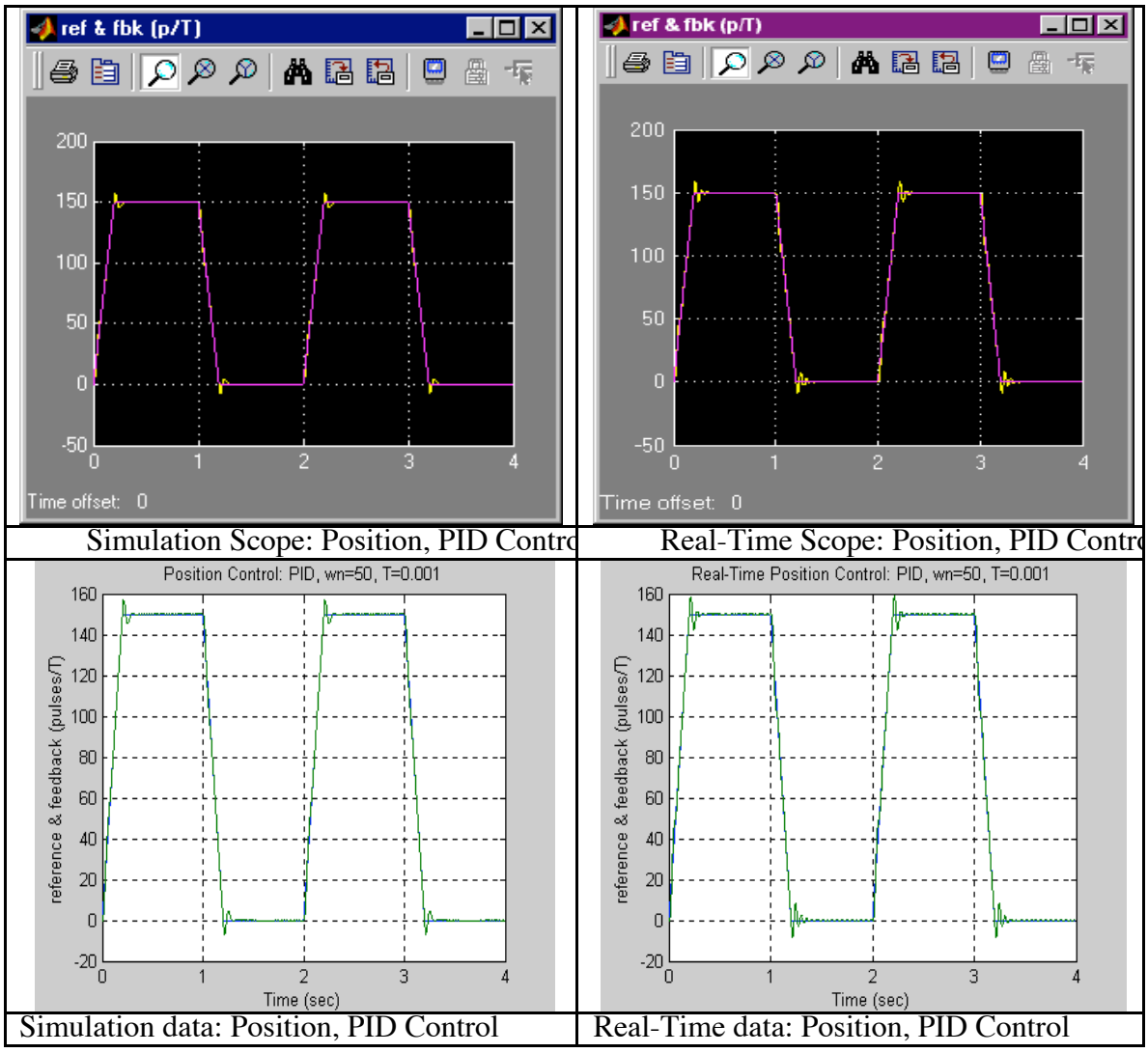


Fig. 5. Simulation (left) and real-time (right) plots of velocity (K_r=0) control; ω_n=100.





Simulation Scope: Position, PID Contro | Real-Time Scope: Position, PID Contro

Simulation data: Position, PID Control | Real-Time data: Position, PID Control

Fig. 6. Simulation (left) and real-time (right) plots of position (K_r=1) control; ω_n=50.