Real-time Actuator Simulator Using Digital Signal Processing

Robert Lynn Mueller
The Pennsylvania State University
New Kensington Campus

Abstract

This paper describes an actuator module that was originally built to convert 4 – 20 mA control signals into stepper motor command signals. The stepper motor is used to position a valve in an airflow process simulator and this actuator module now allows standard PID controllers to be used to control the airflow in the simulator. It is also now possible to use PLCs that don’t have stepper motor commands as part of their PID algorithm. However, by using digital signal processing techniques in the module, it is now possible to simulate actuator delays and time constants. To help appreciate different actuator responses, it is also possible to replace the stepper motor actuator with a pneumatic actuator. This simulation system enables students to modify an actuator’s transfer function without destroying the realism of a hands-on simulator. The paper describes the interface module’s hardware and digital signal processing techniques as well as some of the lab experiments associated with the module.

Introduction

There is no question that the lab is the place to drive home complex control theory. There have been papers suggesting methods [1] – [4] and there are companies that provide equipment [5] – [6]. But sometimes when the equipment is set up, things aren’t quite right for the topic that is being demonstrated. This paper describes a module that can be added to training simulators that can be used to modify the control characteristics to achieve desired results without destroying the realism of hands-on learning. While the module described in this paper is for a flow simulator application, its generalized design makes it easily modifiable for other applications.

The flow simulator is based on one designed by Jim Rehg[4] and is shown in Figure 1. It consists of a vertical tube with a squirrel cage blower at the bottom and a small fan at the top. The fan is used as a generator to measure relative airflow. The damper valve towards the bottom of the tube is used to control the airflow. Either the pneumatic actuator on the left of the tube or the stepper motor on the right side of the tube can control the valve’s position. The actuator not being used must be physically disconnected. The blower’s input is partially blocked so that when the valve is around 90% open, the flow sensor detects approximately 100% relative airflow; this makes the loop gain a little more than 1. This is necessary so that a gain crossover exists when experiments involving phase margin are conducted.
Figure 1. Flow Simulator

A block diagram of simulator system is shown in Figure 2.

"Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition
Copyright © 2003, American Society for Engineering Education"
The actuator module is shown in Figure 3. It accepts inputs of 4 – 20 mA or 0 – 10 Volts and produces outputs of 0 – 5 Volts, 4 – 20 mA, and stepper motor commands. User specified inputs are time delay (also known as dead time or transport lag) in 0.1 second increments up to 9.9 seconds and 2 time constants up to 9.9 seconds. These inputs are set using the thumbwheel switches on the upper right.
Figure 3. Actuator Module

Hardware Implementation

The actuator module is microprocessor-based using the EMAC, Inc’s MicroPac HC11 [7]. It has 20 digital I/O lines, 2 serial ports, LCD interface, keypad interface, 8 channels of A/D and 4 channels of D/A. Since there weren’t enough digital inputs to read all the thumbwheel switches, it was necessary to build a multiplexer. A general hardware schematic diagram is shown in Figure 4 and the multiplexer schematic is shown in Appendix A.
Figure 4. Hardware Diagram of Actuator Module

Software Implementation

The programming language is based on a pretty much a standard BASIC interpreter. It runs on a PC and provides for downloading of the program to the microprocessor. The program listing is included in Appendix B.
When the module is powered up, it initializes some variables and then waits for the Multipurpose Pushbutton to be pressed. When it is pressed, the program moves the stepper motor towards the fully closed position until the pushbutton is pressed again. At this point, it stops the stepper motor and then moves it towards the fully open position until the pushbutton is pressed again. This sequence allows the program to know the relative location of the fully open and closed positions of the valve.

After power-up, the program is a continuous loop that runs every 100 ms and consists of 3 separate subprograms. One subprogram implements the delay function and the other 2 are identical programs that each implement a one-time constant transfer function. Referring to Figure 5, the module’s input is the input to the delay subprogram. The input for the first time-constant subprogram comes from the delay subprogram’s output. The input for the second time-constant subprogram is the first time-constant subprogram’s output and the module’s output is the second time-constant subprogram’s output.

To implement the delay subprogram, a table of length 100 is used to store values from the flow sensor. Every 100 ms the table is shifted down 1 location and the new flow reading is placed in the first table location. If there is no delay being implemented, then the first table location is used for the subprogram’s output. If delay is being implemented, the appropriate table location (index down 1 location is the table for each 100 ms of delay) is used for the subprogram’s output.
output. For example, if the delay is 0.5 seconds, the subprogram’s output would be the table’s 6th entry which corresponds to the flow reading taken 500 ms previously.

The time-constant subprograms are implemented identically. If the subprogram’s input is represented by x and its output is represented by y, then the notation y(n) refers to the subprogram’s output for the current time period, y(n-1) refers to the subprogram’s output for the previous time period, and x(n) refers to the subprogram’s current input. It can be shown [8] that for a 100 ms scan rate that:

\[
x(n) - y(n-1) \\
y(n) = \frac{1}{\text{Weight}} + y(n-1)
\]

The value for Weight is determined from the Time Constant (TC) according to

\[
\text{Weight} = (TC + 0.1) \times 10
\]

Note that a time constant of zero results in the subprogram’s input being passed directly to its output.

**Preparation for Lab Experiments**

For the sake of brevity, only the labs associated with the pneumatic actuator are discussed here.

Before any of the lab experiments can be run, a number of things need to be done. Most of these items have some instructional value and can also be used as experiments themselves.

For the general case, it may be necessary to prepare the hands-on simulator for the interface module. In this case, the simulator was ready for the interface module and it was only necessary to plot valve position versus airflow so that the experiments were run in the linear part of the valve’s operation. This corresponds to a valve position range of 40% to 60% for this simulator.

It may not be necessary but it is instructional to verify that the actuator module is producing the expected results. This is done by placing a square wave generator on the module’s input and examining the output in comparison to the input. This can be done using a PLC trend package, strip chart recorder or similar. Each of the subprograms is verified individually. The delay subprogram is tested by setting the 2 time constants to zero and observing the results with different time delays at various frequencies. Each time constant is done separately by setting the delay to zero and the other time constant to zero. The time constant is the amount of time it takes the output to reach 63% of its final value.
Figure 6. Bode Plots for Flow Simulator

Lab Experiments

In order to run later experiments, it was necessary to produce magnitude and phase Bode plots for the simulator by setting the actuator module’s delay and time constants set to zero. The plots were done for the flow simulator by using the trend package of an Opto 22 PLC. This trend package runs on a PC connected to the PLC and allows for data collection to be saved to the PC’s hard drive in a format that can be used as input data for MS Excel. The PLC was programmed to output a sine wave that could be varied in frequency and this output was then connected to the actuator module’s input. The trend package monitored the sine wave input and the module’s output. The Bode plots made from this data for the pneumatic actuator are shown in Figure 6.

It can also be instructional to have the students obtain a transfer function for the Bode plots. If this is done, then MATLAB can be used to verify experimental results. Examining the magnitude plot in Figure 6 shows that the system has 2 poles in the vicinity of 0.4 radians/sec. Assuming for convenience that they both occur at that frequency gives a loop transfer function (ignoring dead time) of

\[ G(s)H(s) = \frac{1.6}{(s/0.4 + 1)^2} \]

It is now desirable to estimate the delay time (dead time) in the system. It can be shown [10] that the additional phase shift added to a system due to time delay is given by:

\[ \text{Phase (degrees)} \]

\[ -40 \quad -20 \quad 0 \quad 20 \quad 40 \]

\[ w \text{ (r/s)} \]

\[ 0.01 \quad 0.1 \quad 1 \quad 10 \]

\[ \text{Magnitude (dB)} \]

\[ -180 \quad -135 \quad -90 \quad -45 \quad 0 \]

\[ \text{Phase (degrees)} \]

\[ -180 \quad -135 \quad -90 \quad -45 \quad 0 \]

\[ 0 \quad 20 \quad 40 \]

\[ w \text{ (r/s)} \]

\[ 0.01 \quad 0.1 \quad 1 \quad 10 \]

\[ \text{Magnitude (dB)} \]

\[ -180 \quad -135 \quad -90 \quad -45 \quad 0 \]

\[ \text{Phase (degrees)} \]

\[ -180 \quad -135 \quad -90 \quad -45 \quad 0 \]
phase shift = - w* t

Where “w” is the frequency in radians per second and “t” is the time delay in seconds.

A double pole at 0.4 radians/second corresponds to a phase shift of –90 degrees at that frequency. However, the phase Bode plot shows that the phase is –110 degrees. Thus, in order to find the amount of delay that corresponds to the “missing” 20 degrees of phase shift using the above equation:

\[
t = \frac{20 \text{ degrees}}{90 \text{ degrees}} \times \frac{2 \times 3.1416 \text{ radians}}{360 \text{ degrees}} = 0.87 \text{ seconds}
\]

A number of PID-related experiments were run using this simulator. One of them was a comparison of parameters for a PI controller when the actuator’s transfer function is changed. Using the ultimate method proposed by Ziegler and Nichols [9], the following PI parameters were found: Kp = 1.8 and Ti = 5.7 seconds. The step response obtained by changing the input from 20% to 40% for the non-PID-controlled system is shown in Figure 7. The response for the PID-controlled system is shown in Figure 8. As expected, it is noted that the PID-controlled system has no steady-state error and that the response is approximately quarter-wave.

The actuator module is then used to add 2.5 seconds delay into the system. This adds considerable instability to the process. In fact, it was close enough to marginal stability that the results were used to calculate the PI parameters using the Zieler-Nichols method used above. This gave the following parameters: Kp = 0.45 and Ti = 13.3 seconds. The step response to this PID-controller system is shown in Figure 9 and it is noted that the system is now much more stable.

The phase margin concept is easily demonstrated using the actuator module. Referring to the Bode plots shown in Figure 6, the gain crossover frequency is at 0.5 radians/second and the phase at that frequency is –122 degrees; this results in a phase margin of 58 degrees. Recalling the previous equation relating delay time to phase, the amount of delay to negate the phase margin can be found by:

\[
t = \frac{58 \text{ degrees}}{122 \text{ degrees}} \times \frac{2 \times 3.1416 \text{ radians}}{360 \text{ degrees}} = 2.02 \text{ seconds}
\]
Thus, setting the actuator module for 2.0 seconds of delay should remove the entire phase margin. However, the actual amount of delay for this system turned out to be a little more than 2.5 seconds. Giving this system a step response to this system does in fact demonstrate that the system is marginally stable. The effect of delay can be further demonstrated by reducing the delay to make the system more stable and by increasing it to make the system more unstable.
Any experiments involving the time constant aspect of the module must have time constants faster than the hands-on simulator being used otherwise the simulator will provide the dominant characteristics of the overall system and the actuator module will not any significant effect.

Experiments involving damping ratios and break frequencies can be interesting. For example, setting the delay to zero and the 2 time constants to 9.9 gives a system with a break frequency of approximately 0.1 Hertz and a damping ratio of 1. Giving this system a step change results in an output of the classical critically damped system. Other experiments using just one time constant to add enough negative phase shift to cause marginal stability can also be instructional.
Conclusion:

This paper has described an actuator module that allows the transfer function of an actuator to be modified so that lab experiments can more closely demonstrate desired results without materially detracting from the realism of a hands-on. It has also described a few lab experiments that have been implemented using a flow simulator.

Acknowledgements:

The author would like to thank newly graduated engineer Steve Elliott for his assistance in the construction of the actuator module and in data collection.

Bibliography:

“Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition
Copyright © 2003, American Society for Engineering Education”


ROBERT LYNN MUELLER
Robert Lynn (Doc) Mueller is an assistant professor of Electro-Mechanical Engineering at the New Kensington Campus of the Pennsylvania State University. He received his B.S. in Electrical Engineering from Wichita State University and his Ph.D. in Electrical Engineering from the University of Pittsburgh. He is a registered professional engineer in the state of Pennsylvania and a member of several professional societies. Dr. Mueller spent 30 years working with industrial control systems before joining Penn State and still continues to work as a consultant providing advice regarding industrial automation.
Appendix A
Detailed Hardware Schematics
Appendix B
Program Listing

1 x=$FF
2 poke($8001,X)
3 y=$00
4 poke($8009,y)
5 DIM DD(50)
7 N1=0
8 N2=0
9 for i=0 to 49
10 DD(i)=0
11 next i

25 REM -----------Start 100ms Loop---------

30 x=$7F
40 poke($e000,x)
41 ZZ=portd
42 ZX=(ZZ.and.$FC)/4
51 ZX=ZX*10
52 DT=ZX

60 x=$BF
70 poke($e000,x)
80 ZZ=portd
90 ZX=(ZZ.and.$FC)/4
100 DT=DT+ZX

110 x=$DF
120 poke($e000,x)
130 ZZ=portd
140 ZX=(ZZ.and.$FC)/4
150 ZX=ZX*10
151 TC=ZX

160 x=$EF
170 poke($e000,x)
180 ZZ=portd
190 ZX=(ZZ.and.$FC)/4
200 TC=TC+ZX

210 x=$F7
220 poke($e000,x)
230 ZZ=portd
240 ZX=(ZZ.and.$FC)/4
250 ZX=ZX*10
251 TB=ZX

260 x=$FB
270 poke($e000,x)
280 ZZ=portd
290 ZX=(ZZ.and.$FC)/4
300 TB=TB+ZX
301 IN=adc(0)
302 i=49
303 for l=1 to 49
304 j=i-1
305 DD(i)=DD(j)
306 i=i-1
308 next l
309 DD(0)=IN
310 W1=TC+1
320 W2=TB+1
325 U=DD(DT)*100
330 O1=N1
340 N1=((U-O1)/W1)+O1
350 O2=N2
360 N2=((N1-O2)/W2)+O2
365 Q=N2/100
390 goto 30