

A CLOSED-LOOP TEMPERATURE CONTROL SYSTEM BY UTILIZING A LABVIEW CUSTOMER-DESIGN PID CONTROLLER

MOHAMMAD A.K. ALIA, MOHAMMAD K. ABU ZALATA

*Faculty of Engineering Technology
Al-Balqa' Applied University
Amman, Jordan
mohammed_abudaih@yahoo.com*

Abstract By using LabVIEW (G language), a PID control algorithm was simulated. The designed virtual instrument includes all the necessary components and facilities necessary to function properly and to control any linear process. The issues of integral windup, derivative overrun, output signal limits and sampling interval Δt were treated. The created Sub VI was used as a controller in a closed loop which controls an oven temperature. Results of experiment show a high degree of convergence with the results obtained when an electronic hard-wired controller was utilized.

Keywords PID algorithm, Sampling period, Integral windup, Soft-ware PID controller, Hard-wired controller.

INTRODUCTION

During the period from the mid-fifties to the mid-seventies, the analog computer was widely used to obtain the response of control systems. During that time digital computing was very costly and slow compared to the situation today. There was little soft-ware available, and one had to program the solution to a problem using machine code. During the last (15) years, the picture has been drastically changing for the welfare of digital computers. As the cost of digital computing decreased and its speed of operation increased, the analog computer was gradually replaced by the digital computer. Nowadays, to study the effect of control strategies and controller parameters on the response of a complex control system, one must use a computer simulation. One of the generic control strategies is the PID control algorithm. If PID controller is properly tuned, it will produce an acceptable control for most industrial processes. The PID controller represents the ultimate in control of a continuous process for which a specific mathematical description (transfer function) cannot be written [Curtis D. Johnson 1984].

To program the PID algorithm using a higher-level language, such as C, Pascal, or Fortran, the work will be much easier and less error prone than when using assembly language. High level languages allow us to use floating-point math. Negative numbers and overflows are handled automatically [J. Michael Jacob, 1989]. An excellent representative of high level languages is LabVIEW. Merits of this language are given in detail in [National Instruments, 1996], [National Instruments, 2002], [Barry E. Paton 1999]. We think that the major advantage of LabVIEW over conventional high level

languages is the graphical user interface, which is built in, intuitive in operation, and simple to apply. According to [National Instruments, 2002] productivity is better with LabVIEW than with conventional languages for a factor of (5-10) times compared with C on a small project.

If one does not have PID simulation software, as in our case, it is necessary to write one's own computer program. It is still instructive to learn how to write computer programs (Virtual instruments- VIs) for the purpose of understanding the problems and limitations associated with commercially available simulation software. Examples of such problems are selecting the step size of the independent variable (Δt) or establishing initial conditions [Donald R. Coughanowr,19917]. In his valuable resource [National Instruments, 2002], Gary W. Johnson writes:"I wrote the PID VIs in the LabVIEW PID control Toolkit with the goal that they should be easy to apply and modify. Every control engineer has personal preferences as to which flavor of PID algorithm should be used in any particular situation. You can rewrite the supplied PID functions to incorporate your favorite algorithm. Just because I programmed this particular set (which I personally trust) doesn't mean it's always the best for every application".

Building on the above, we were enthusiastically encouraged to try to simulate the PID algorithm in our own way by using the basic LabVIEW functions, and to verify the simulated algorithm by testing it practically in a temperature control system. Experimental results show that when the designed PID software and when an electronic hard-wired PID controller were used the responses of

temperature control system for a unit-step input, are practically the same.

Another advantage of computer-based software control is the flexibility available in modifying control strategy. This makes it possible to do further improvements to achieve better system performance. For example, it is not difficult to modify the PID control algorithm by changing the gain during the transient period in order to minimize system overshoot and response time, keeping at the same time a zero steady-state offset. Hereinafter, we shall describe the control algorithm and illustrate the experimental results.

Programming PID Algorithm

The PID controller VI consists of four Sub VIs: proportional, integral, derivative and Δt Sub VIs. The mathematical algorithm of PID controller is as follows:

$$v_{out}(t) = K_p e(t) + K_I \int e(t) dt + K_d \frac{de(t)}{dt} \quad (1)$$

where $e(t)$ is the error. K_p , K_I , K_d are coefficients of proportional, integral, and derivative actions respectively.

Proportional VI

For proportional controller only, the following equation applies:

$$P = K_p e(t) \quad (2)$$

In order to get the proportional action simply multiply the error $e(t)$ by the constant of proportionality (gain) K_p . the block diagram is shown in Figure (1).

Integral VI

The integral action is evaluated by using the trapezoidal method. The mathematical representation of the integral action is as follows, (Figure (2)).

$$\int_0^n e(t).dt = \sum_{k=0}^n \left(\frac{e(kT) + e[(k+1).T]}{2} \right) \cdot T \quad (3)$$

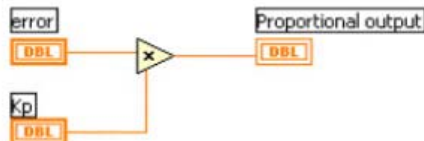


Figure (1) Block Diagram of Proportional Action VI

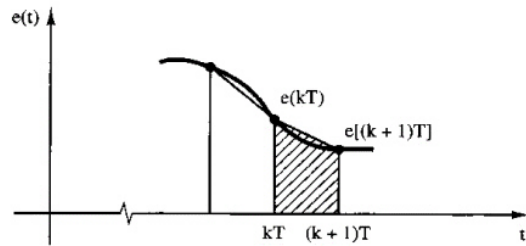


Figure (2) Trapezoidal Method of Integration

where $K=0, 1, 2, \dots, n$, and $T = \Delta t =$ sampling rate. The block diagram of the integral VI is shown in Figure (3).

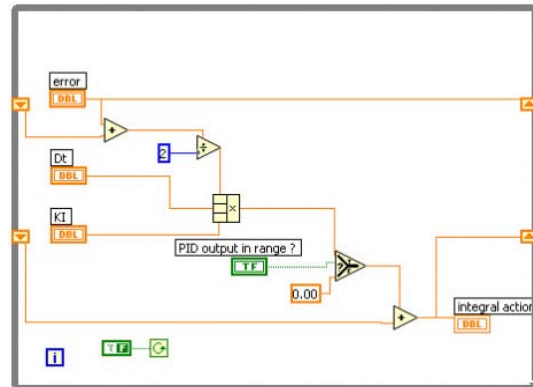


Figure (3) Block Diagram of Integral Action VI.

When the integral action VI is called, the while loop executes once because the conditional terminal is connected to a Boolean constant (false). The error passes through the **error** control. The error value is added to the value of the error from the past iteration. This is done by the shift register. Then the value of the summation is divided by 2 and multiplied by the gain K_I and the sample time Δt . After that, a Boolean value passes through the Boolean control **PID in range?**. This is to investigate whether the **PID** output limits are reached or not. If the Boolean value is true, this means that the **PID** output lies within its limits, and the calculated value of integration is added to the previous values. If the Boolean value is false then the **PID** is saturated and the value at the output of the integral action must stay constant until the **PID** output is within the range again.

Derivative VI

The mathematical equation of the derivative action is the following:

$$v_{o,d} = K_d \frac{de(t)}{dt} \quad (4)$$

The derivative action will be calculated by using the backward difference method, Figure (4):

$$\frac{de(t)}{dt} = \frac{e(KT) - e[(K-1).T]}{T}$$

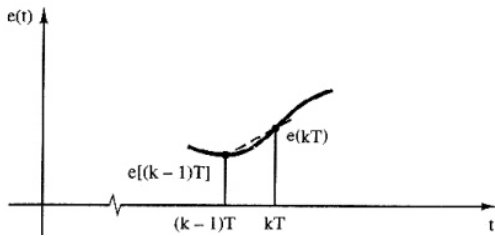


Figure (4) Backward Difference Method

The block diagram of derivative VI is shown in Figure (5). The process variable passes to the VI through the P_v control. The past value of the process variable is subtracted from the current P_v value, then the result is divided by the sample time Δt , and the final result, after division, is multiplied with the derivative gain (K_d). Using the select function, a condition to prevent the division by zero (in case that Δt was zero) is included. If Δt is larger than zero, the calculated value passes out to the **Derivative action** indicator. If Δt is less or equal to zero, for some reasons, then a value of zero is passed to the **Derivative action** indicator.

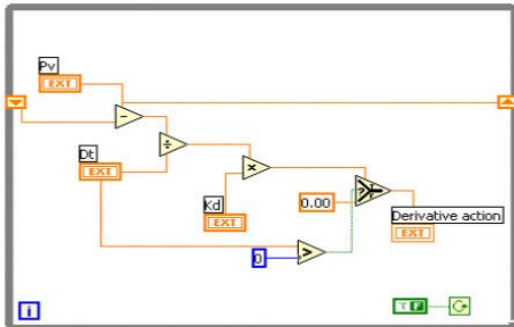


Figure (5) Block Diagram of Derivative VI.

Δt Virtual Instrument

This VI is necessary to calculate the time period between every two samples (Δt). This VI is very important because it is needed for real time control. It has only one output and has no input. The block diagram is shown in Figure (6). The tick count function returns the internal timer of the CPU in milliseconds. The reference time is undefined, so the tick count value cannot be converted to real world time. The timer value warps from $(2^{32}-1)$ to zero. In the first iteration of the loop the tick count function gives the internal timer value, then the initial value stored in the shift register (which is equal to zero in the first iteration) is

subtracted from this value. The result is the time in millisecond elapsed between the first call and the second call of this VI. After dividing this value by 1000, the result is in seconds.

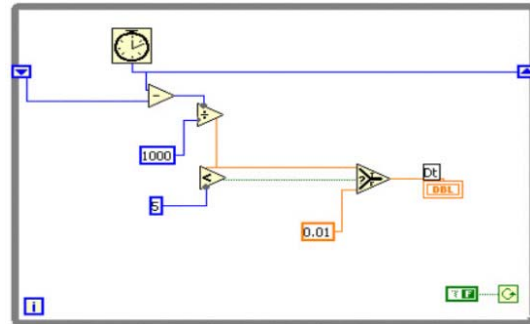


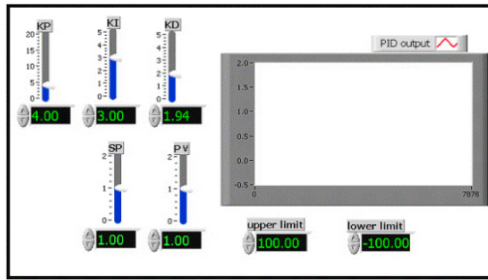
Figure (6) Block Diagram of Δt VI

In the first call for this VI, the time difference between the first iteration and the previous one is the internal timer value. Because this value is not referenced to a known value, it is very large and is not a true time value, so, it must not be passed out. For this purpose a condition statement was developed by comparing this value to an arbitrary number which represents the maximum delay allowed. If this value is larger than the arbitrary number, then it passes a number 0.01, that is close to the real value in the following iterations. If the value is smaller than this arbitrary number, then the VI passes the value, which is the right time difference. In the next iteration, the value of the tick count gives the present internal timer value. By subtracting the previous value stored in the shift register from the present value of the tick count, we get the time elapsed between this iteration and the past iteration, which is the time needed to execute the whole program once. In other words, it is the time between the first sample and the next sample.

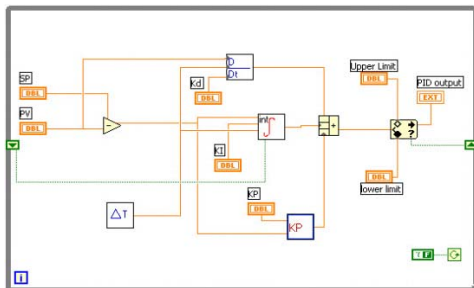
Complete PID VI

The PID output is the sum of the outputs of the proportional, integral and derivative actions. The front panel and block diagram of the PID VI are shown in Figure (7). The values of process variable (P_v) and the error $e(t)$ are passed to the PID VI through the P_v and **error** controls. The output of the PID passes out through the **PID output** indicator. If the output of the PID is within the range between **upper limit** and **lower limit** controls, then this value passes out at **Coerced(x)** terminal and a true Boolean value at **In Range?** terminal of the **In Range and Coerce** function. If the output of the **PID VI** is larger than the **upper limit**, then the value of the **upper limit** passes out and False passes to the shift register. But if the output of the **PID VI** is less than the **lower limit**, the value of the **lower limit** passes out and False value passes to shift register.

Here, the Boolean value is used to prevent the integral wind up.



(a)



(b)

**Figure (7) a- PID VI Front Panel
b- PID VI Block Diagram**

Temperature Control by Using PID Virtual Instrument

The temperature control system is a laboratory experiment board manufactured by DELLRENZO-Italy [Gary W.Johnson 1994]

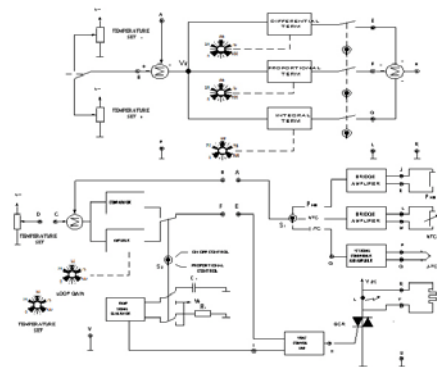


Figure (8) RGT1 Board

The board allows to experimentally investigate both the temperature measurements and closed loop temperature control. The board includes an electric oven model, temperature transducers and an electronic hard-wired PID controller. For comparison purposes, oven temperature was controlled by the simulated PID (VI) and the built in

electronic PID controller. The principal circuit of the experiment board is given in Figure (8). The front panel and the block diagram of the temperature control VI are shown in Figure (9).

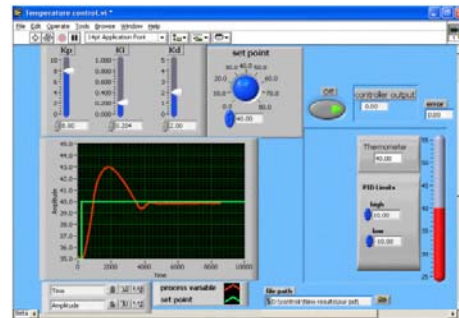
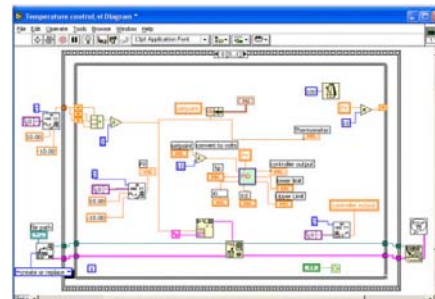
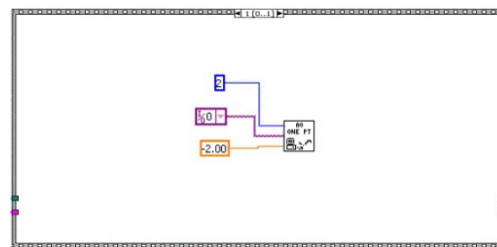


Figure (9) a- Temperature Control VI Front Panel



(b)



(c)

**Figure (9)
b-Temperature Control VI Block Diagram
c- Sequence 1**

Results of Experiments

1- Proportional Only Control Mode by Using PID VI. Figure (10) shows system response for three different values of system gain. By increasing the gain the offset decreases, and the time rise decreases also.

2- P, PI, and PID Control, When Using PID VI. Figure (11) shows system responses for the three modes of control.

3- System Responses, When It Was Exited from the Load-Side, and by Changing the Set point, (input side) System responses are shown in Figure (12)

4- Comparison Between the Hard-Ware and Soft-Ware Controlled System.

Figure (13) illustrates the achieved responses for P, PI, modes of control.

CONCLUSIONS

1. By using basic LabVIEW controls, indicators and block diagram functions, a graphical PID controller was designed and tested.

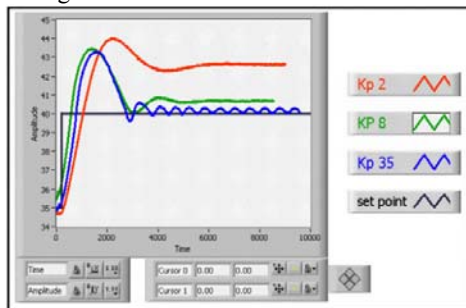


Figure (10) Proportional Control where $K_p = 2, 8, \text{ and } 35$

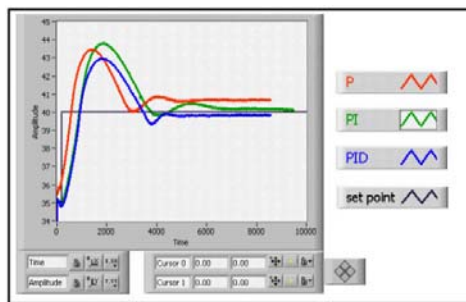
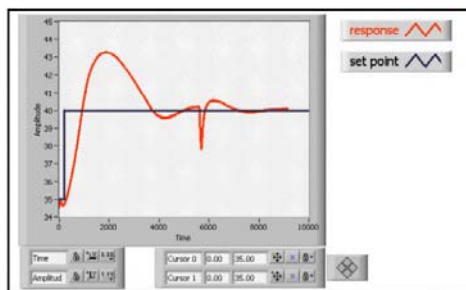
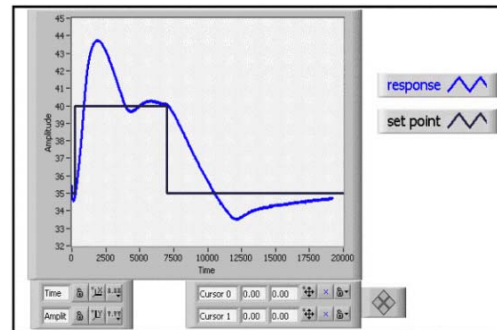


Figure (11) System Response to P, PI, PID Controllers.



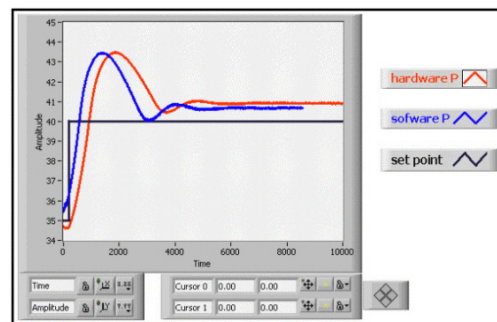
(a)



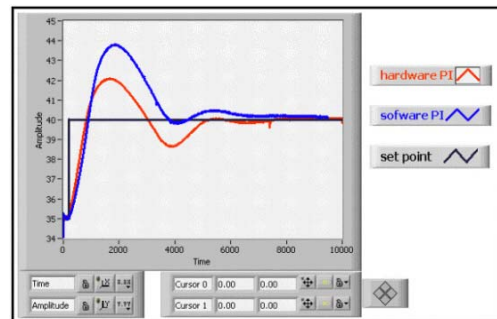
(b)

Figure (12)

a- System Response to Load Disturbance
b- System Response to Change in Set Point



(a)



(b)

Figure (13): a- Response to Hardware and Software P Controller; b- Response to Hardware and Software PI Controller

Creating a Sub VI which resembles the built electronic controller makes it easier to tune the controller through variation of the controls K_p , K_i , and K_d , without referring to the internal components of the PID block diagram.

2. By using the front panel, simulation of the PID algorithm gives the user flexibility to observe in real time the variation of any input, output and intermediate system variable, and to carry out on-line excitation from the load side or from the input side.

3. When realizing software PID controller, it is easy to include the necessary additional modifications and enhancements such as integral windup, derivative overrun, controller output limits, and selection of the sampling interval

4. Experimental results show that there is a high degree of convergence between the performance of the software PID controller and the performance of the electronic hard-wired PID controller, when both were utilized in closed-loop temperature control.

REFERENCES

Barry E. Paton 1999," LabVIEW Graphical Programming for Instrumentation". Prentice Hall PTP, New Jersey , U.S.A

Curtis D. Johnson 1984," Microprocessor-Based Process Control". Prentice-Hall International, Inc Englewood Cliffs, NJ , U.S.A.

DE-LORENZO, 2002 ,Electronic Laboratory," (Basic Board To Study Temperature Regulation)". DL.2155RGT1, D1.2155RGT2, Milano, Italy

Donald R. Coughanowr,1991,"Process Systems Analysis and Control". McGraw-Hill, Inc., Singapore

Gary W. Johnson, 1994. "LabVIEW Graphical Programming". McGraw-Hill, Inc., New york, U.S.A

J. Michael Jacob, 1989," Industrial Control Electronics". Prentice-Hall International, Inc., New Jersey , U.S.A

National Instruments, 2002, LabVIEW Basics Introduction, Course Manual, U.S.A

National Instruments, 1996, "LabVIEW Graphical Programming for Instrumentation". User Manual. New Jersey , U.S.A