Implementation of an Autotune Variation (ATV) Control Algorithm to Expand an Undergraduate Process Control Laboratory

By **Brian Seamans** and **Peter Rony**
Department of Chemical Engineering
Virginia Tech
Blacksburg, VA 24060-0211

Introduction

In the Spring 2000 issue of CACHE News (Spring, 2000), we described in detail the newly renovated, junior-level process controls laboratory at the Department of Chemical Engineering, Virginia Tech. Our objectives for the article were to (a) introduce you to the advantages of Wonderware InControl and InTouch software and (b) offer assistance to you (once you obtain a Wonderware educational license).

The Virginia Tech process control laboratory consists of six experiments -- each linked via an ethernet network -- that are controlled by Pentium® personal computers running Wonderware® Intouch and InControl application software. The laboratory represents a control atmosphere similar to that which graduates may encounter in commercial plants. The objectives of the lab course are to expose students to the control of real-life unit operations on a small scale and to develop the students' understanding of process controls. In the limited range of a lab, diversity of experiments and control strategies is important to cultivate a student's comprehension and appreciation of the subject matter.

The upgrading of the ChE 3016 lab experiments continues at a vigorous pace. With the Wonderware InControl program and InTouch HMI developed by Robert Rice [1] as a base, Brian Seamans and Justin Wagner have recently added three Advanced Process Control algorithms, namely the Autotune Variation Technique (ATV), the Oscillogenic Instrument, and a modest Statistical Quality Control algorithm, respectively. All three programs have run successfully during the spring 2001 process controls laboratory. In this article, we wish to describe the Wonderware implementation of the ATV algorithm.

The spring, Wonderware-based, ChE 3016 undergraduate process controls laboratory has now operated successfully for two years (spring 2000 and spring 2001). The Wonderware HMI (human/machine interface) has operated with few problems, and has permitted student groups to reduce experimentation time from approximately 3 hours (pre-2000 laboratories) to 2.5 hours or less per lab session.

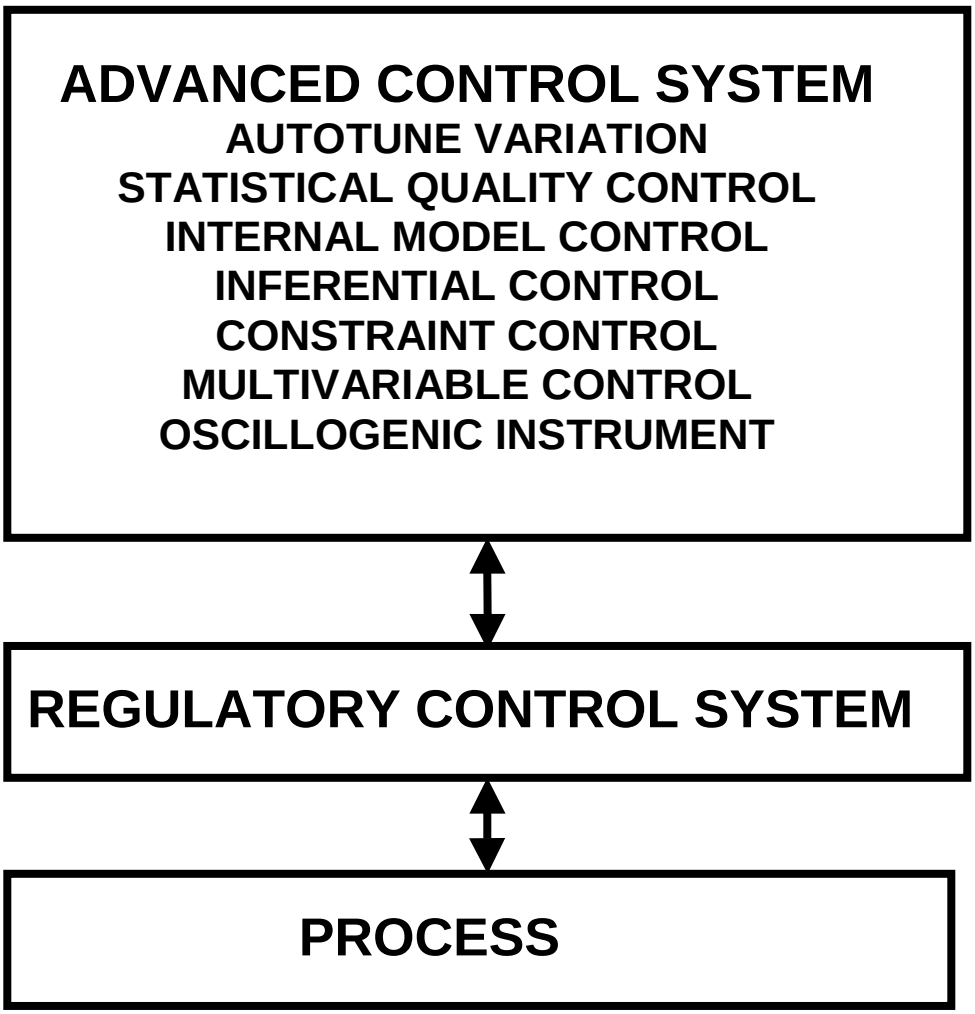

Figure 1. Three lower levels of control automation. This figure is based upon a presentation by Sanjay Mansukhani to chemical engineering seniors at Virginia Tech [2].

Autotune Variation Technique (ATV): The Theory of Relay Response Tuning

Several methodologies exist for autotune proportional/integral (PI) controllers. One scheme is to create a "relay response" in which a hardwired relay circuit is placed in series with the controller and apparatus. The pulsing of this circuit creates a square wave in the manipulated variable of the system (MV) and a corresponding oscillating response in the process variable (PV). The oscillating response seen in the PV simulates the process under Ziegler-Nichols ultimate control conditions, that is, it is an undamped oscillation, from which measurements of both ultimate controller gain (Ku) and ultimate period (Pu) can be made, as seen in Figure 2. The ultimate gain is calculated according to Equation (1) [3]:

$$K_u = \frac{4h}{\pi a} \tag{1}$$

These calculated values, when substituted into the Ziegler-Nichols online controller tuning equations, result in useable controller parameters, such as the proportional gain and integral time of a PI controller. The usefulness of this method lies in the ability to obtain controller parameters from a running process with a minimal disturbance to the system. A software-based controller also displays the ability to create

square wave to simulate an ultimate response without requiring a hardwired relay [3]. A simple program, it is either executed or ignored as to the operator's desires.

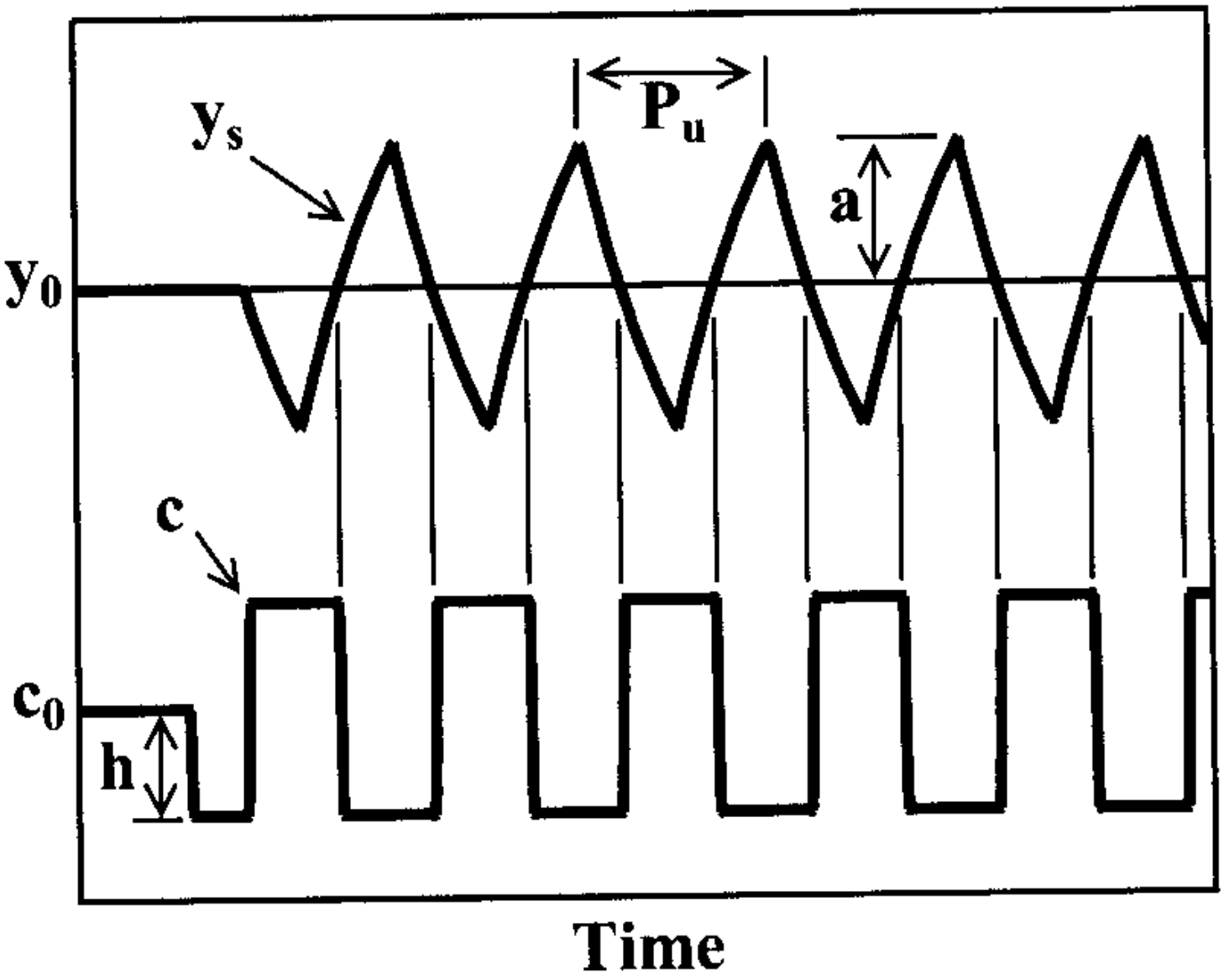

Figure 2. Graphical representation response of a system undergoing an ATV test. The parameters, y and c, are the process and manipulated variables, respectively. This figure taken from the textbook, "Chemical Process Control", by James B. Riggs [4].

Development of a Wonderware-Based Autotune Variation Algorithm

The precedent for an Autotune Variation experiment in the Virginia Tech process controls lab existed because of the work of David Ingram [5], who programmed such an experiment using Texas Instruments' Application Productivity Tool (APT) software. The experiment became outdated, however, because of the limitations of the DOS-based APT software during the 1990s. Once the entire lab received an overhaul to Wonderware® applications software, the uniformity and programmability of the new software allowed for the development of a Wonderware-based Autotune Variation experiment. The idea behind the experiment was to create a program in Wonderware structured-text language (STL) that would create a

square-wave action in the manipulated variable. The resulting data would then be used to calculate both the ultimate gain and ultimate period of the process.

To achieve this oscillation, the lead author created an ATV program (see Appendices A and B).. Whenever the PV moves above the setpoint, the manipulated variable reduces to the lower bound, causing a decrease in the PV. The opposite occurs when the PV moves below the setpoint. This continual change of the manipulated variable, creates the desired mock ultimate response in the PV.

Once the program creates the oscillating response, it then needs to calculate both the ultimate gain and ultimate period of theclosed-loop system. For the ultimate gain, the program maintains two arrays, one for the high end of the cycle and one for the low. The program evaluates each real-time data point from the field device and stores the highest or lowest value of a cycle in the next available slot, moving all previous inputs back one space. Each array holds the previous five peaks or valley cycle points. The difference between each maximum and minimum pair is twice the ultimate gain. The ultimate gain is calculated in real time as a moving average of the last five cycles. This averaging system lessens the effect of any single-cycle disturbance from skewing the ultimate gain. Similar to the ultimate gain calculation, the ultimate period relies on a five-point moving average. The program has two internal timers that record the time at which the set point is at its upper and lower values. These two timer values for each cycle are used to determine the ultimate period.

It is left to student groups to use a textbook table of Zeigeler-Nichols online tuning parameters in order to manually calculate quarter-decay-ratio proportional-gain and integral-time values.

After the program was completed and tested, the need arose for a human/machine interface (HMI) to enable junior chemical engineering students to run the auto-tuning program on each of the six lab experiments. The requirements for the HMI were to (a) turn the program on and off, and (b) display the resulting calculated ultimate gain and ultimate period of the process. Figures 3 and 4 show the HMIs developed to regulate the operation of the autotune program. The HMI appears as a secondary window within the computer environment that becomes active by a selectable "Autotune" button on the main Wonderware screen.

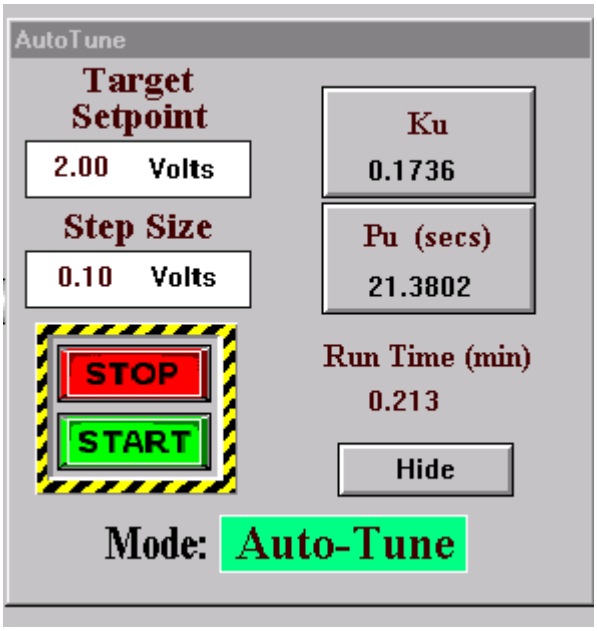

Figure 3. The HMI window developed to run the autotune program. This is the pop-up window that appears when a student clicks on an Autotune button on the main HMI screen. The students first enter both the target setpoint and also the step size, then click on START. After one lab period, we concluded that the window parameters and algorithm were not too useful. With very little reprogramming, this pop-up window and its associated algorithm were changed to the window shown in Figure 4.

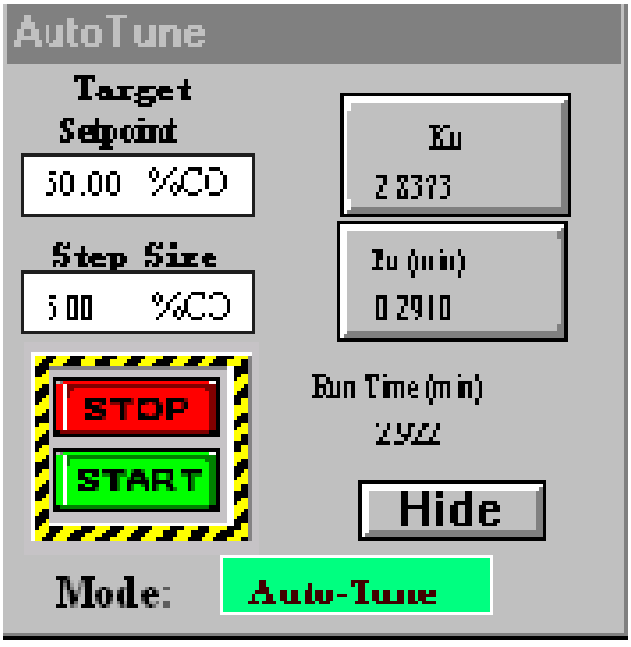

Figure 4. The revised HMI window developed to run the autotune program. In this case, students enter the target manipulated variable, in %, and the step size, in %. NOTE: We forgot to change the word, "setpoint" to "operating point."

In addition to the minimum requirements, the HMI also includes input fields for a target operating point and a step size along with a display of the duration of the experiment execution. The target input allows a student to customize the autotune run to a specific operating point. This enables the student to create controller parameters around the same operating point at which they preformed an initial step change, resulting in comparable parameters for the controller. The ability to specify a step size means that a student can also test the best size step to produce reasonable Autotune results. The read-out of the calculated ultimate gain and period are not only important for the final results of the experiment, but also add a means of qualifying a steady-state oscillations. Some experiments have the capacity to exhibit a relay response with very small steps. Such small steps coupled with the resolution of the real-time data trends may make it hard to observe a steady oscillation pattern. However when the real-time ultimate gain and period calculations start to show only small changes over time, it is a safe assumption that the oscillating pattern of the auto-tune program closely resemble the desired ultimate response.

After completing the autotune algorithm and the HMI, one interlock was added to the autotune algorithm to prevent unnecessary execution. An Interlock is an element of a program that, either for safety or resource control, must be met before a program will run. The autotune algorithm can only operate when the software-based controller is in automatic control. This interlock prevents the program from using valuable processor time when it is not capable of executing correctly.

Results

In order to develop and test the autotune program, the lead author used the operation amplifier (OPAMP-1) experiment, which consists of a quad opamp integrated-circuit chip wired as a cascaded fourth-order system, with each first-order system having a time constant of 2 seconds. This experiment displayed a total time constant of about 0.1 minutes and an almost linear operating curve. The simplicity of this apparatus and its quick response made it ideal for a case study. Since the only variable of the equipment was the voltage that propagated through the first-order systems, there were no external disturbances that factored into the recorded response of the system. This meant that only the autotune algorithm and the associated Wonderware program influenced the behavior of the OPAMP-1 system. After the program was able to create the relay response, it was straightforward to test the newly suggested controller parameters against alternatively derived IMC PI tuning parameters.

First, the lead author created an open-loop operating curve for the OPAMP-1 experiment. Analysis of this curve yielded an operating point of 2.0 volts. By creating data files [2] for a step change around this operating point, IMC PI controller parameters were recommended from a first-order plus dead-time (FOPDT) model using Doug Cooper's Control Station software [7]. A closed-loop response curve was performed on the system by increasing the setpoint from 1.5 to 2.5 Volts. Analysis of the closed-loop data via Control Station yielded a closed loop gain of 0.98.

Second, with the controller in automatic, the lead author ran the autotune program to obtain both the ultimate gain and ultimate period. The desired oscillating pattern was clearly evident.

With the ultimate gain and period values in hand, the lead author converted them into a controller gain (Kc) by the equation $Kc = 0.45 * Ku$, and an integral time (Ti) by $Ti = 0.85 * Pu$. These equations were based on the Ziegler-Nichols online tuning method. The new controller parameters were entered into the PI controller and the closed-loop point change test rerun. Based upon the Control Station analysis, the closed loop gain equaled 1.001 for the new tuning parameters.

Tables I and II provide OPAMP-1 data taken on April 5, 2001 by the student group of Brooke Lynch, Jilian Staniec, Sara Marlow, Mandy Graham, and Sara Sarver.

Table I. Open Loop Model parameters and initial IMC PI tuning parameters

Model Parameters:	
Kp (%PV/%CO)	1.04 ± 0.21
Tp (min)	0.109 ± 0.011
Td (min)	0.069 ± 0.028
IMC PI Tuning:	
Kc (%CO/%PV)	0.84 ± 0.17
Ti (min)	0.11 ± 0.011

Note: The closed loop gain obtained from Control Station for these initial IMC PI tuning parameters was 1.01 ± 0.20

Table II. Autotune Results for Varying Step Sizes

Run	**Step Size (% CO)**	**Ku (%CO/%PV)**	**Pu (minutes)**	**Kc (%CO/%PV)**	**Ti (minutes)**	**Closed Loop Gain K**
1	2	2.6562 ± 0.2	0.2913 ± 0.08	1.21 ± 0.08	0.243 ± 0.275	0.95 ± 0.19
2	1	2.1775 ± 0.2	0.3153 ± 0.08	0.99 ± 0.09	0.263 ± 0.254	0.91 ± 0.18
3	0.5	2.1280 ± 0.2	0.2847 ± 0.08	0.97 ± 0.09	0.237 ± 0.281	0.95 ± 0.19
4	0.25	2.2293 ± 0.2	0.1303 ± 0.08	1.01 ± 0.09	0.109 ± 0.614	1.01 ± 0.20

Note: Run 4 seems to give the "best" parameters for control because the closed loop gain is almost 1.0

The reproducibility of the ultimate gain was tested repeatedly. It was not possible to achieve similar results on the OPAMP-1 system for open-loop response curve step sizes as small as one-tenth of one percent of the overall range, or 0.05 volts. This very small change in the process demonstrates how useful an autotune experimental system can be.

Finally after thorough testing the ATV algorithm on the **OPAMP-1** experiment, the algorithm was exported to the remaining lab experiments: **HEAT**, **FLOW**, **PRESS**, **VORTEX**, and **PYRO**. Please see Figures 11 through 17 for the respective HMI interfaces. Testing of such systems supported the ability to export a generic STL autotune program to any system. All six of the laboratory experiments ran the same program to create an autotune relay response and subsequent values of the ultimate gain and ultimate period. In most cases the suggested autotune-based parameters were an improvement to the originally calculated open-loop step-change values. Only PYRO gave problems, primarily because of a poor initial choice of step size.

Autotune Experiment Procedure for Student Groups

The procedure for a student's autotune experiment follows closely the procedure used by the lead author for testing the autotune algorithm.

- Create an open loop operating curve and select an operating point.
- Create an open loop response curve around the selected operating point.
- Analyze the open-loop step test data in order to obtain controller parameters based upon Control Station Design Tools software [7].
- Using IMC PI Control Station tuning parameters, run a closed-loop step test with the controller in automatic.
- Determine the closed-loop gain and time constant..
- With the controller in automatic, run the autotune program to obtain new controller tuning parameters based upon Ziegler-Nichols tuning equations.
- Using the Ziegler-Nichols tuning parameters for a PI controller, perform the same closed-loop step test as before and determine the closed-loop gain and time constant.
- Variations could include changing the target setpoint and step size in the autotune program.

The Six Autotune Experiments and associated Autotune HMIs

Digital camera photographs for each of six the different experiments -- FLOW, HEAT, OPAMP-1, PRESS, PYRO, and VORTEX -- are shown in Figures 5 through 11, respectively. With an online manuscript, we have the luxury of including such color images without the need to pay for expensive, printed, color reproductions.

Screen captures for the corresponding human/machine interface (HMI) for each of six different experiments are shown in Figures 12 through 17, respectively. The nature of the experiments are summarized as follows:

FLOW -- Control of flowing water through a 0.75-inch-diameter pipe. The water flows through a large, air-to-close control valve. The flow rate is measured using a differential pressure (DP) cell. An I/P transducer is used to provide pneumatic pressure to the control valve. This experiment, created by senior ChE student Steve Thiel in 1981, has worked flawlessly for 19 years until the DP cell gasket developed a leak last year. The gasket was repaired using rubberf cement in spring 2001. (Figure 5).

HEAT -- Control of the temperature of air that flows through a glass, packed-bed containing glass beads and steel wool. The final control element is a heating coil from a vintage-1981-hair dryer. This robust,

M.S. thesis experiment -- created by Barry Archer, a M.S. ChE student, has worked flawlessly for 20 years. (Figure 6).

OPAMP-1 -- Control of the output voltage from a fourth-order BIFET operational amplifier circuit . The time constant of each cascaded, first-order circuit can be varied from 2 to 7 seconds. This robust and portable experiment was created in 1984 by Riley T. Chan, a technician in the ChE department. (Figure 7).

PRESS -- Control of the air pressure in a medium-size vessel. This open system contains an I/P transducer, an $58 Omega differential pressure sensor, and an air-to-open control valve. This experiment was designed by a team of four outstanding freshmen engineering students -- J.C. Taliaferro, Doug Plante, Wes Marner, and Jonathan Thomas -- in fulfillment of their freshmen design project in 1994. (Figure 8).

PYRO -- The Pyroluminescent Regulometer™ apparatus created by Dow Chemical Corporation as a teaching tool for their Camile data acquisition and control system. This $2000 apparatus consists of a light bulb, a small fan, a cylindrical Lucite column, a pair of thermocouples, and an electronic interface box that contains, among other devices, a triac-based, solid-state switch. The light bulb provides a wonderful visualization of the changes in the manipulated variable. (Figures 9 and 10).

VORTEX -- The apparatus consists of a Vortec® vortex tube along with an V/P transducer, an air-to-open valve, and a thermocouple. This experiment was designed by a senior ChE student, Jonathan Witt, during the late 1980s. (Figure 11).

Of forty-six students who responded to the following question -- Which of the experimental systems was most interesting to you? -- in a May 2001 controls-lab survey, the voting was VORTEX (17 votes), PYRO (9 votes), HEAT (8 votes), OPAMP-1 (6 votes), and PRESS (6 votes).

It should be noted that an incorrect pop-up window (Figure 3) appears in Figure 12. Figures 13, 14, and 17 contain the correct pop-up windows (Figure 4).

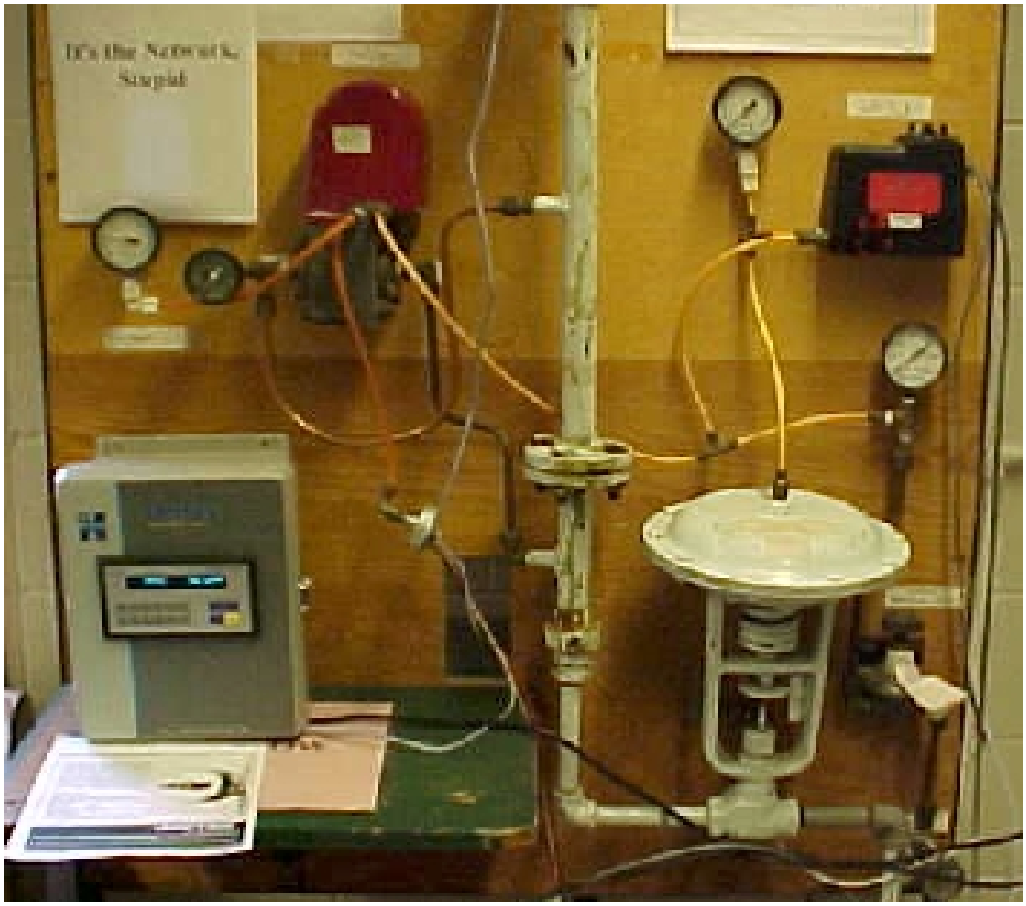

Figure 5. The FLOW experiment, which consists of an air-to-close valve, an I/P transducer, an orifice plate, a DP cell, and a P/V transducer. An electronic flowmeter is in the process of being installed.

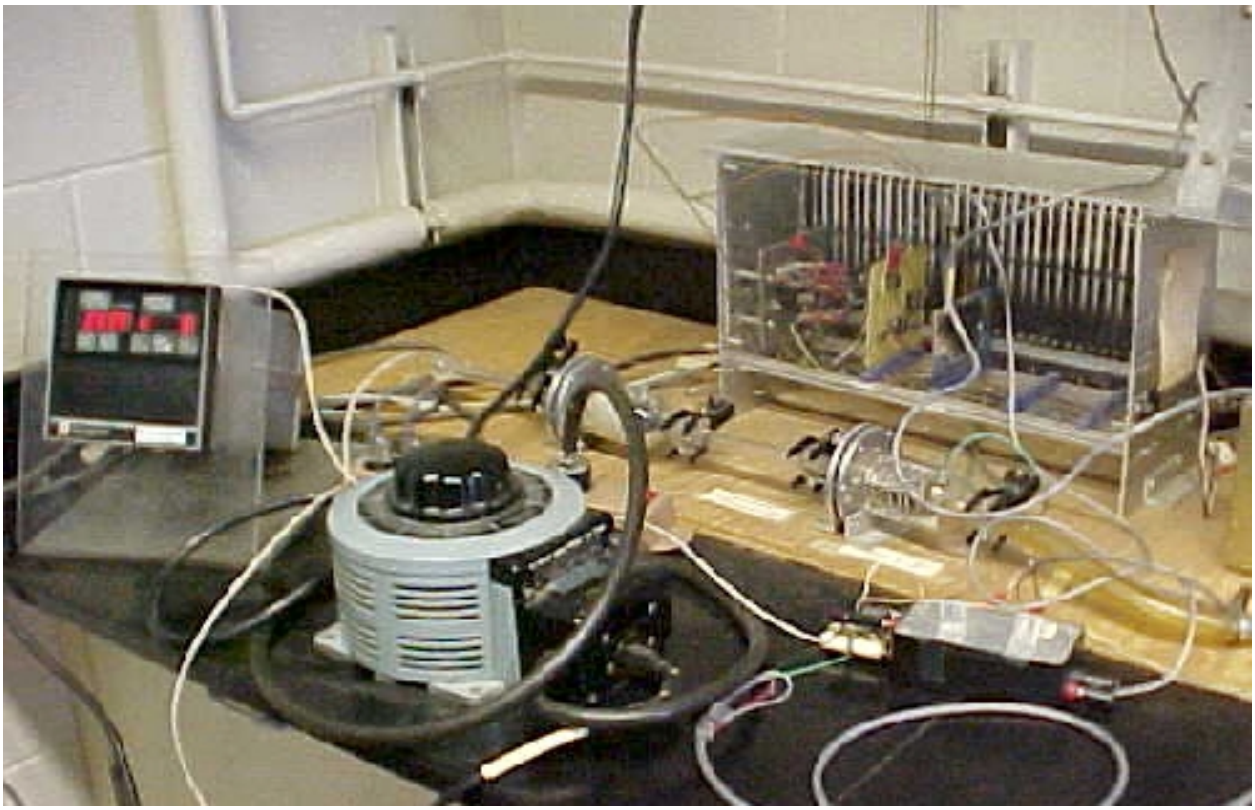

Figure 6. The HEAT experiment, which consists of a heating coil from a hair dryer, a glass bed containing glass beads, a thermocouple, and interface electronics (contained in the rack).

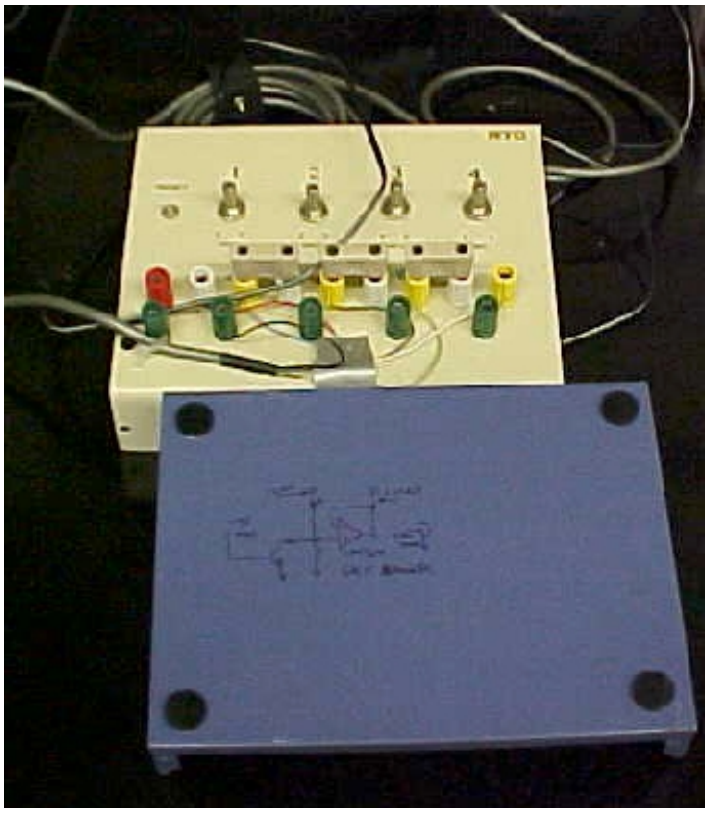

Figure 7. The OPAMP-1 experiment, which consists of a four, cascaded, operational amplifiers from a BIFET TL084 integrated circuit. The time constant of each first-order op amp circuit can be varied between 2 and 7 seconds.

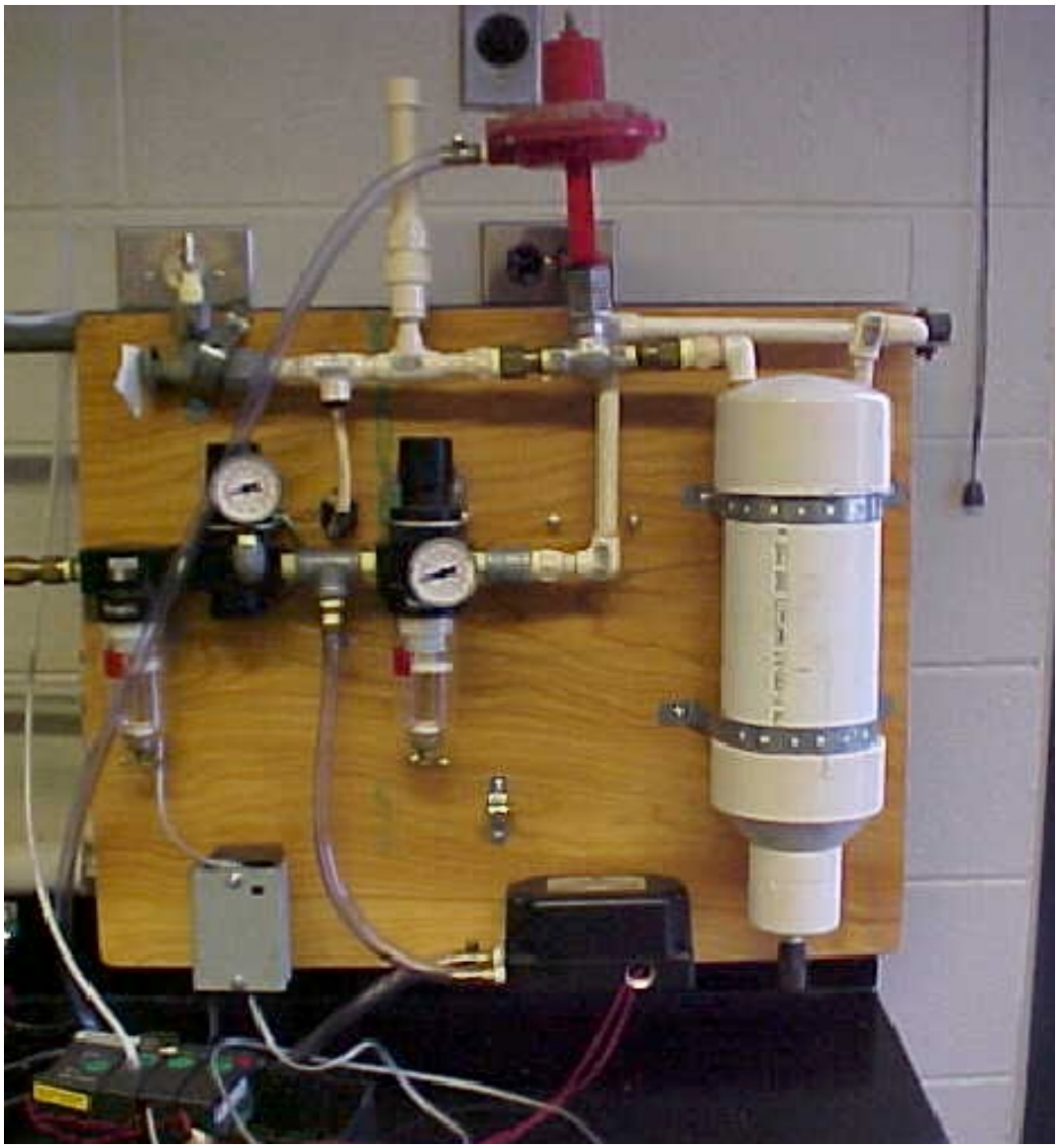

Figure 8. The PRESS experiment, which consists of a ballast container, a V/P transducer, an Omega pressure transducer, and an air-to-open control valve.

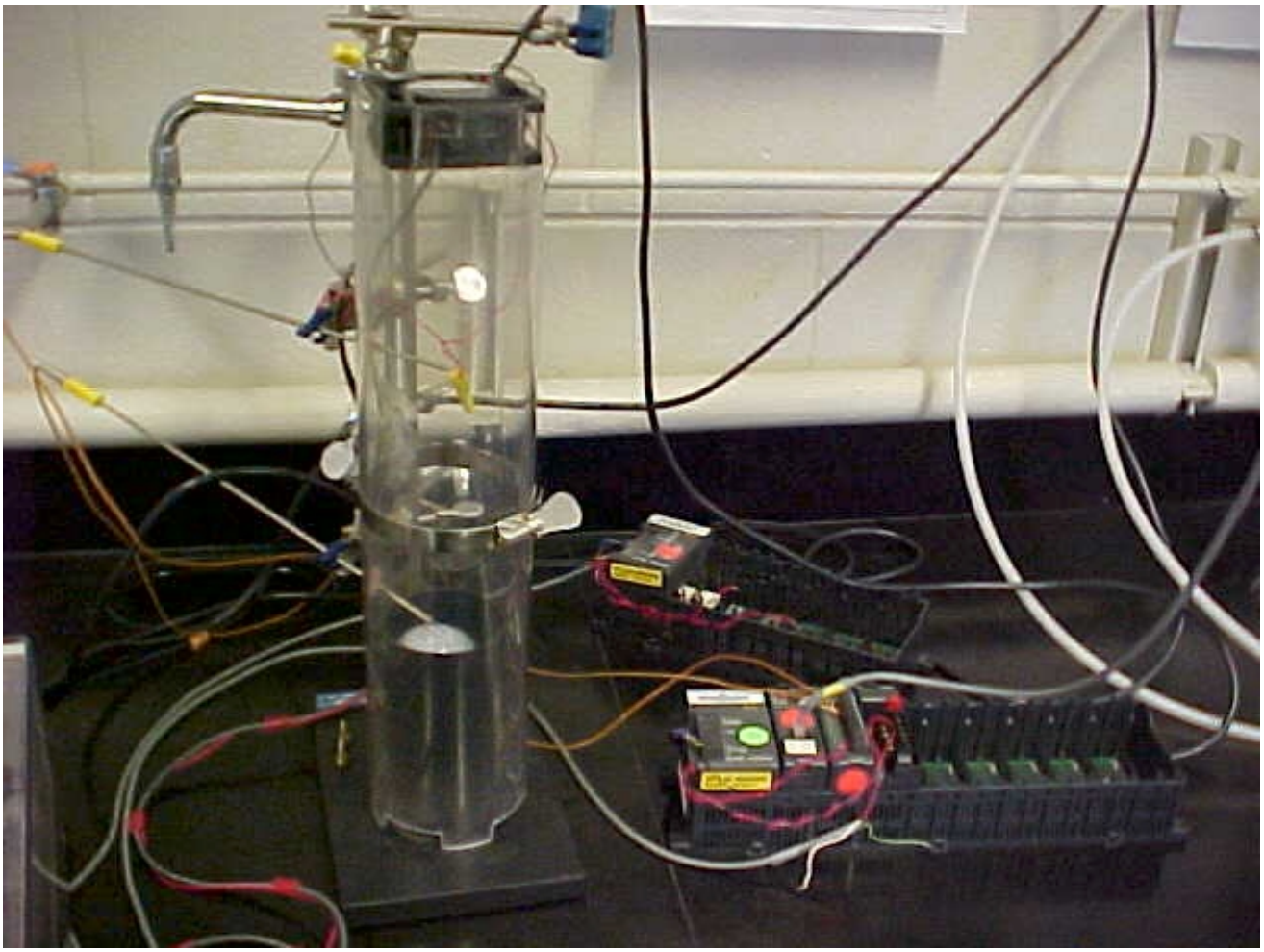

Figure 9. The PYROLUMINESCENT REGULOMETER (PYRO) experiment, which consists of a light bulb, a small fan, a cylindrical Lucite column created by Dow Chemical Co. Also shown is the black, AutomationDirect.COM I/O backplane, which contains a power supply, an Ethernet module, a thermocouple module, and a D/A module.

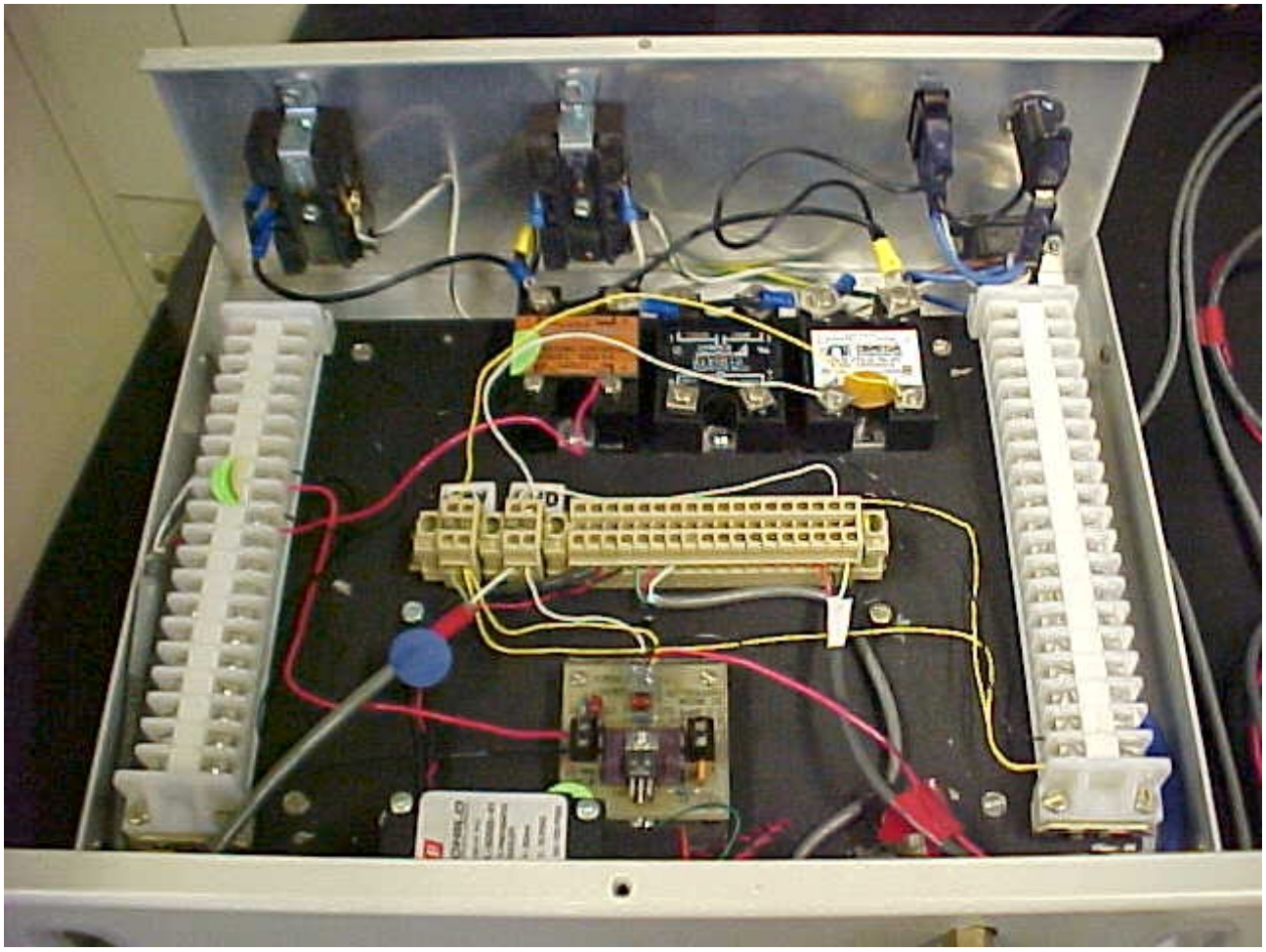

Figure 10. The PYRO experiment interface electronics created by Dow Chemical Co.

Figure 11. The VORTEX experiment Vortec® vorrtex tube, which is insulated.

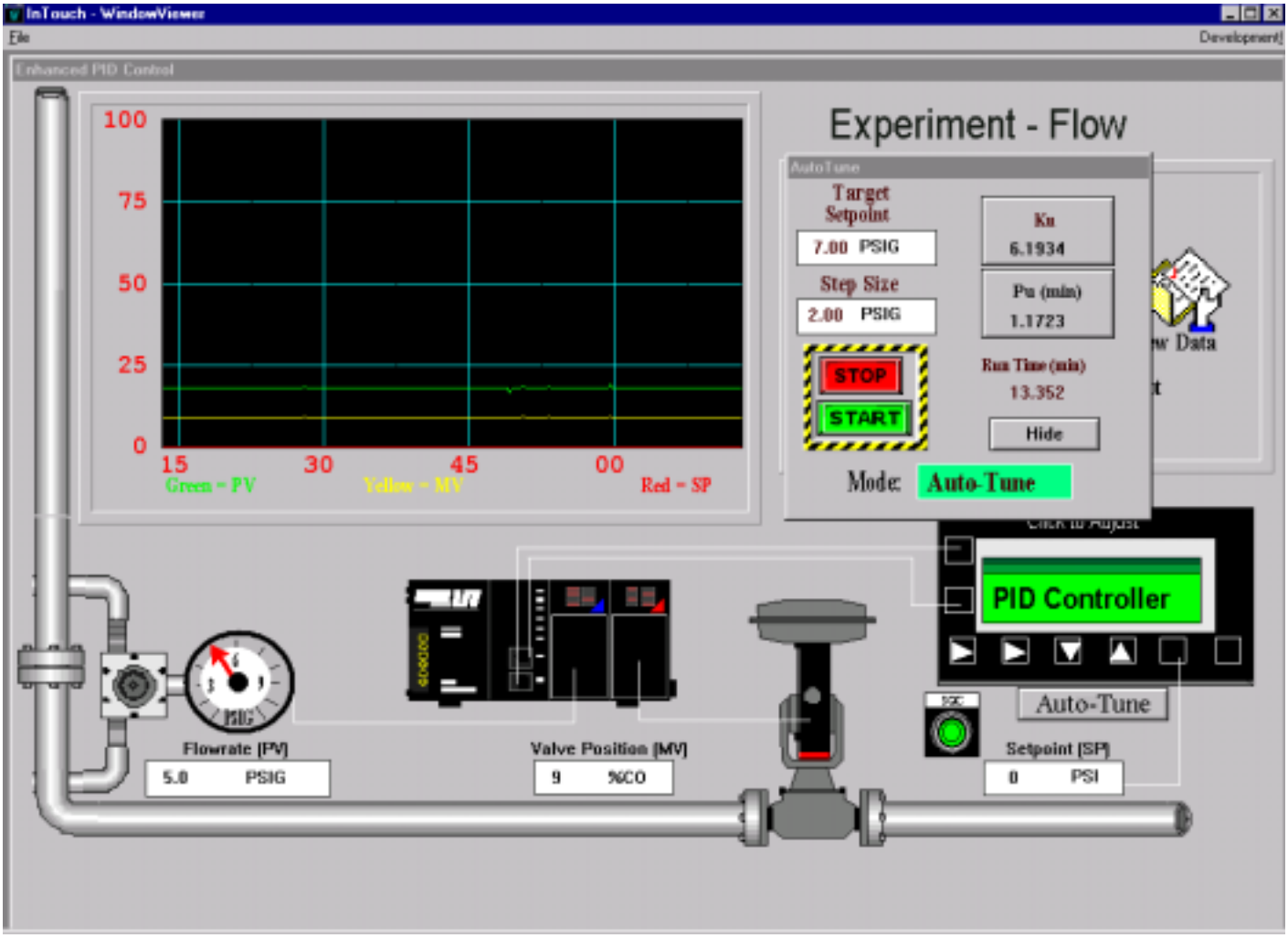

Figure 12. The HMI for the FLOW experiment.

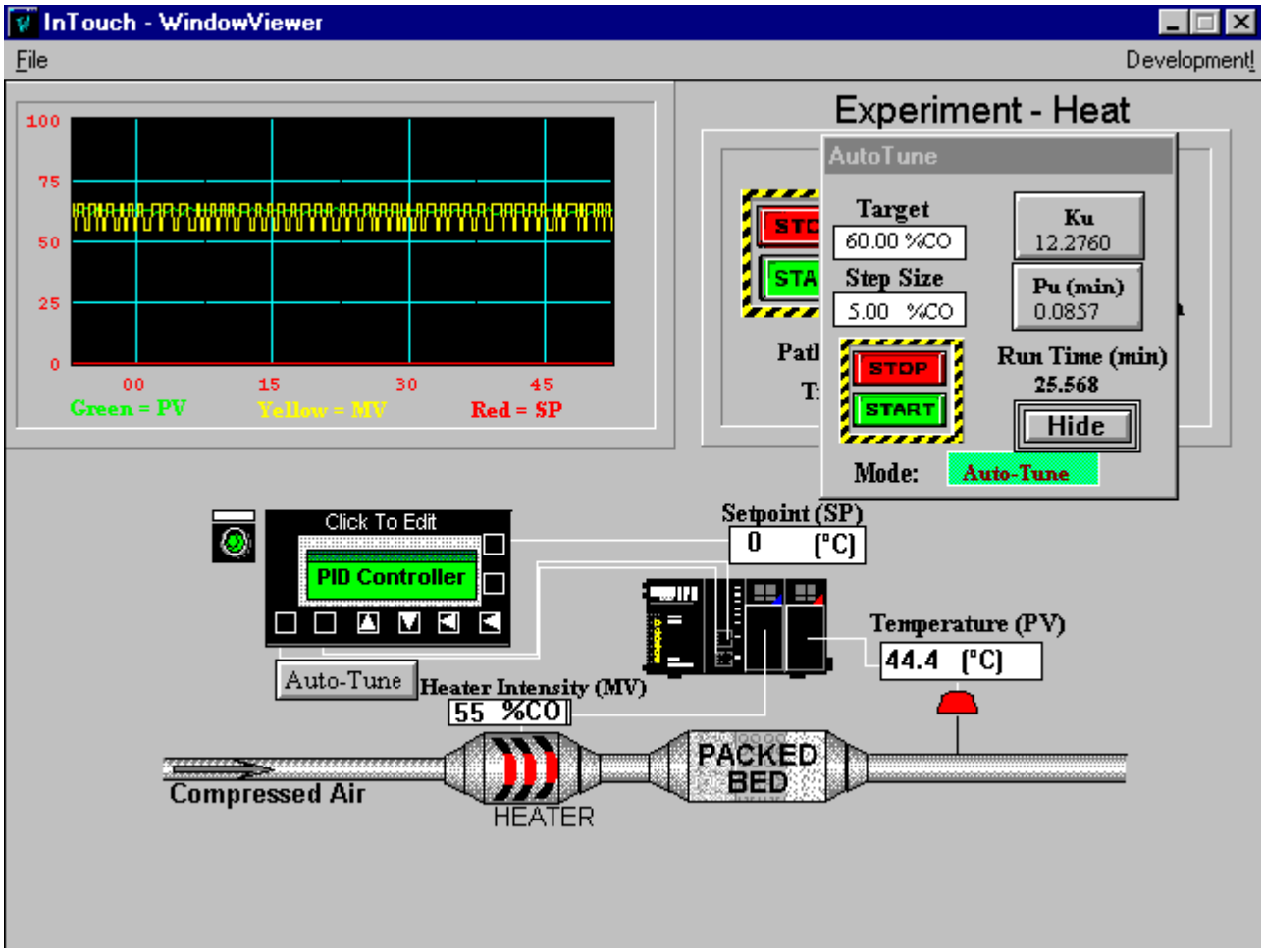

Figure 13. The HMI for the HEAT experiment.

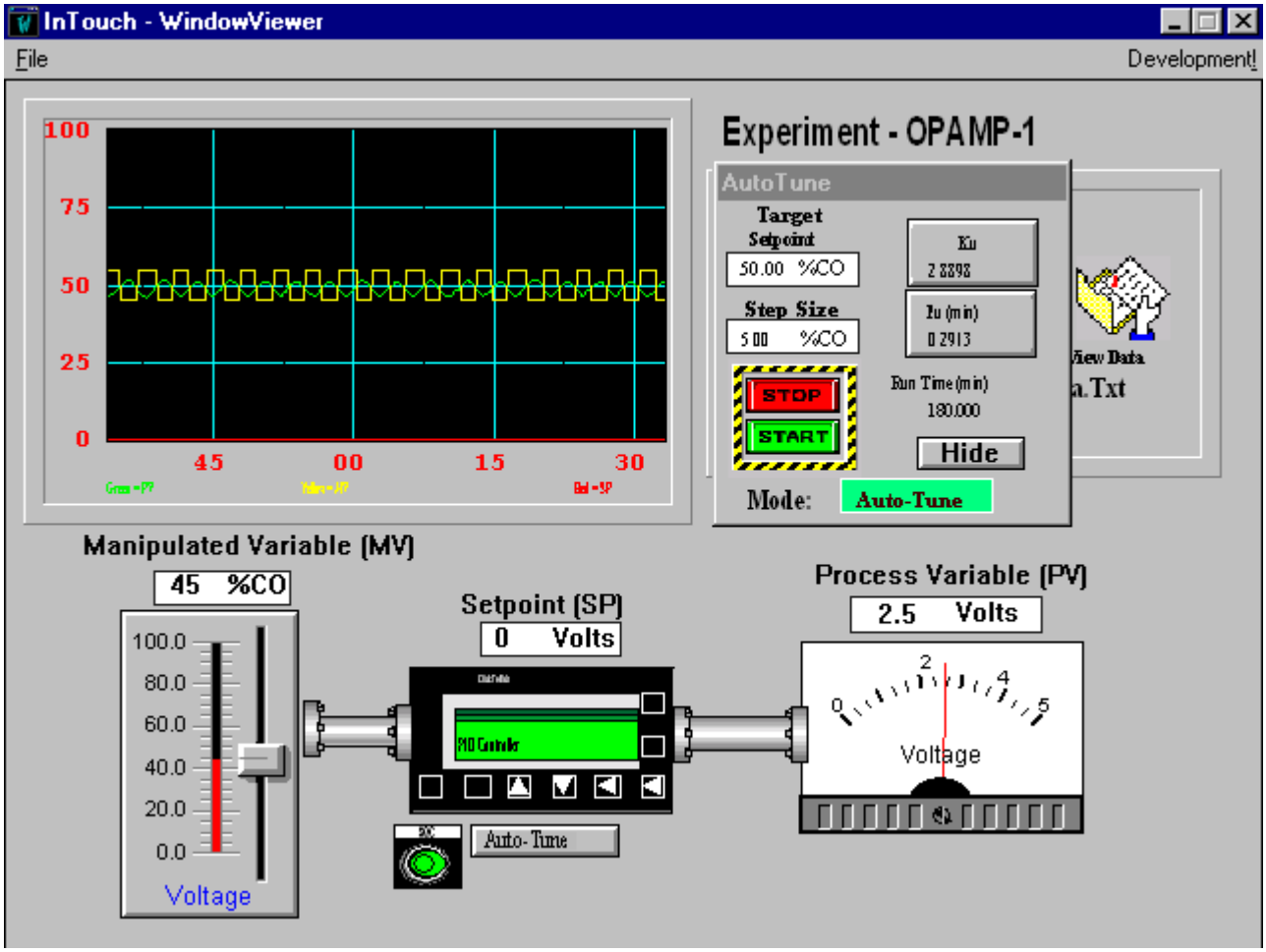

Figure 14. The HMI for the OPAMP-1 experiment.

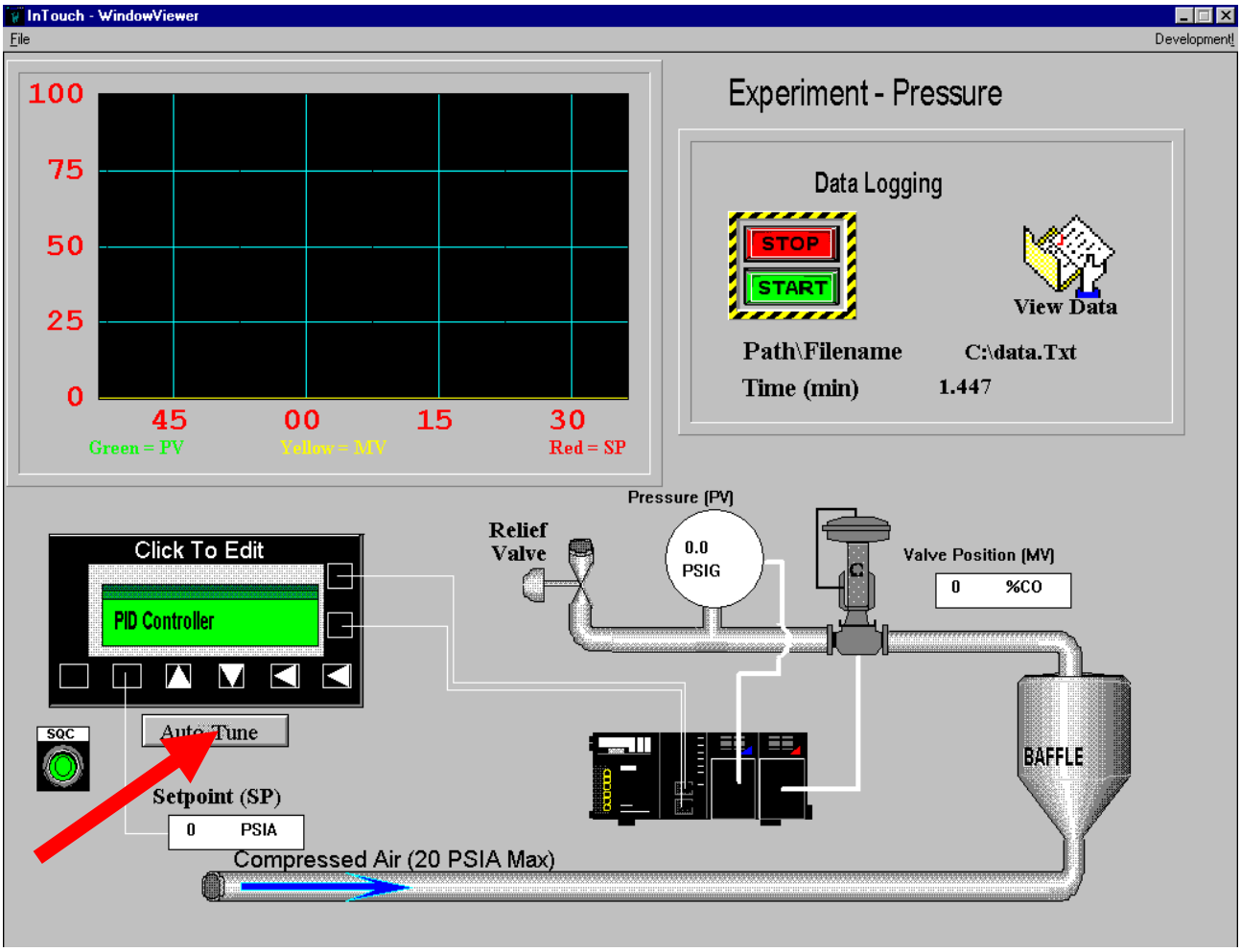

Figure 15. The HMI for the PRESS experiment.

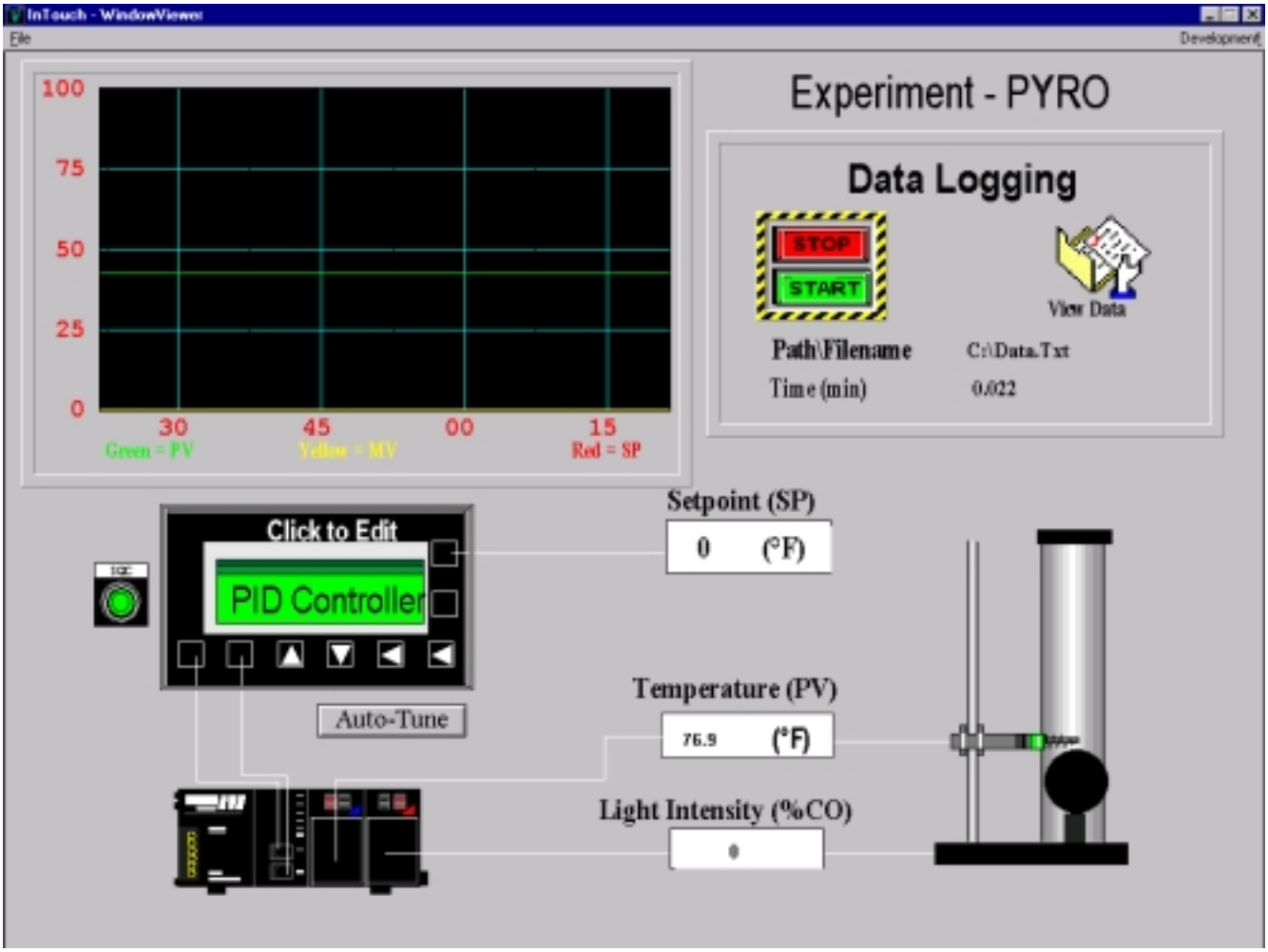

Figure 16. The HMI for the PYROLUMINESCENT REGULOMETER (PYRO) experiment.

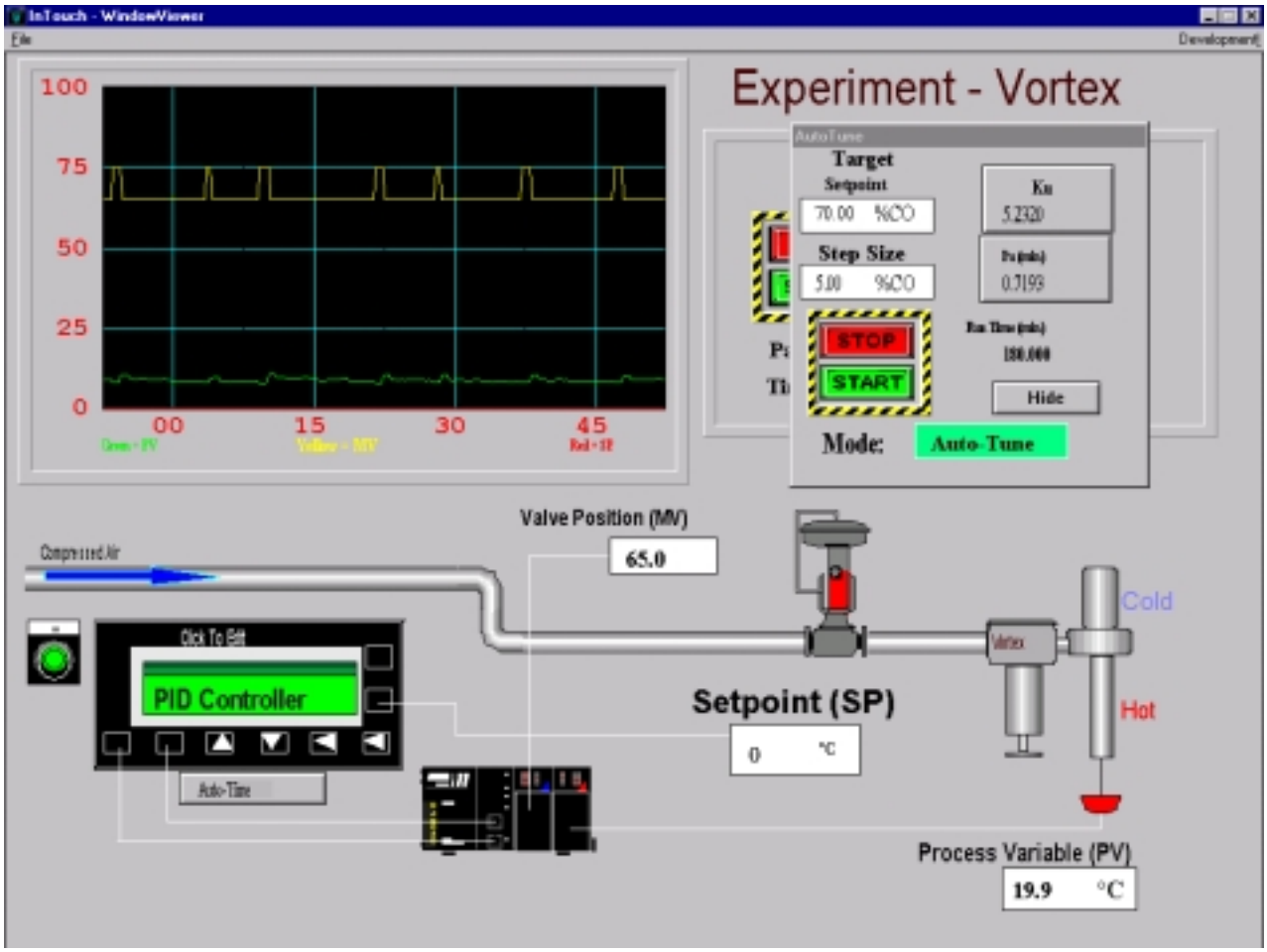

Figure 17. The HMI for the VORTEX experiment.

Glossary

a	Half of peak-to-peak PV peak height
Cycle	A term describing one high-low set of the ultimate response, one period
FOPDT	First order plus dead time model, used to fit responses of first order systems
h	Step height for manipulated variable
Kc	The gain of a controller (here, a PI controller)
Ku	The ultimate gain of a process
Pu	The ultimate period of a process
Ti	The integral time of a controller (here, a PI controller)

References

1. Peter Rony, Robert Rice, Jim Robertello, and Karl Rony, "Enhancing the Process Control Laboratory with Generic, Multivendor Hardware, Software, and a Network, CACHE News, No. 50, pp. 17-27 (Spring 2000).

2. Sanjay Mansukhani, "Cyclohexanone Distillation Optimization using Advanced Process Control and Six Sigma Methodology", Virginia Tech Seminar, February 2001.

3. K. J. Astrom and T. Hagglund, "Automatic Tuning of PID Controllers", Instrument Society of America (1948), p. 233.

4. James B. Riggs, "Chemical Process Control", Ferret Publishing [2609 24th Street, Lubbock, Texas 79410], 1999, p. 197.

5. David Ingram and Colin Runac, “”Heat Transfer Process Automation”, Fall 1991.

6. Peter Rony, Robert Rice, and Jim Robertello, “Enhancing the Process Control Laboratory with Generic Multivendor Hardware, Software, and a Network.”, CACHE.News (No. 50), p. 17-27 (Spring 2000).

7. Doug Cooper, “Control Station”. See http://www.engr.uconn.edu/workbook.html.

8. A print-out of the complete autotune program and table of all variables (Table I) is included in the appendix.

APPENDIX A: Table I

Table I. List of variables used in the autotune program

Tag Name	Type	Status	Description
Auto_Tune_Stop	Bool	Global	Boolean bit that stops autotune progr
Auto_Tune_Start	Bool	Global	Boolean bit that starts autotune progr
Auto_Tune_Running	Bool	Global	Boolean bit that shows status of autotune p
Run_timer	Timer	Local	Timer that counts the time the autotune prog
Elasped_Time	Real	Local	A real number interpretation of the time from
Cycle_Hi	Real	Local	The highest value of the upper bound c
Cycle_Low	Real	Local	The lowest value of the lower bound c
Max_PV	Real Array(5)	Local	Array of last five cycle max PV readi
Min_PV	Real Array(5)	Local	Array of last five cycle minimum PV rea
Period1	Real Array(5)	Local	Array of last five upper bound time
Period2	Real Array(5)	Local	Array of last five lower bound time
Target_Sp	Real	Local	The target mid-line point of the autotune prog
Range	Real	Local	The step size of the autotune program (
Auto_Time1	Timer	Local	Individual upper bound cycle time
Auto_Time2	Timer	Local	Individual lower bound cycle time
Avg_Max	Real	Local	The average value of the 5 Max_PV va
Avg_Min	Real	Local	The average of the 5 Min_PV value
Period	Real	Local	The average period of the autotune prog
Amplitude	Real	Local	The amplitude of the cyclic respons
Ku	Real	Local	The ultimate gain
Pu	Real	Local	The ultimate period
Clear	Int	Local	Integer to reset arrays
Count	Int	Local	Integer to calculate the averages

APPENDIX B: Autotune Algorithm

```
(*Autotune logic for experiments*)

If ( Auto_Tune_Stop ) then
   Auto_Tune_Start:=False;
   Auto_Tune_Running:=False;
   Run_Timer.En:=False;
   Elapsed_Time:=(time_to_real(Run_Timer.Et))/60.0;
   PIDControl.Out:=User_Target;
   stable:=false;
   pvinit:=0;
   pvsecond:=100;
   delay.En:=false;
   delay1.EN:=false;
   one_shot:=true;
   Auto_Tune_Stop:=False;
   end_if;

If ( Auto_Tune_Start ) then
    Auto_Tune_Stop:= False;
    Run_Timer.Pt:=real_to_time(10800);
    Cycle_Hi:=PIDControl.PvEuLo;
    Cycle_Low:=PIDControl.PvEuHi;
    clear:=0;
    (*delay.Pt:=real_to_time(30);
    delay1.Pt:=real_to_time(30);*)
    for clear:= 1 to 5 do
       Max_Pv[clear]:=PIDControl.PvEuLo;
       Min_Pv[clear]:=PIDControl.PvEuHi;
       Period1[clear]:=0.0;
       Period2[clear]:=0.0;
    end_for;
    PIDControl.Out:=User_Target;
    (*If NOT (stable) and (one_shot) or (delay1.Q) then
     pvinit:=PIDControl.Pv;
     delay.EN:=true;
     one_shot:=false;
     delay1.Q:=false;
    end_if;
    If NOT(stable) and (delay.Q) then
     pvsecond:=PIDControl.Pv;
     delay1.En:=true;
     delay.Q:=false;
    end_if;
    If (ABS((pvinit - pvsecond))<=5)then*)
    Target_Sp:=PIDControl.Pv;
    Range:=User_Range;
    Auto_time1.Pt:=real_to_time(3600);
    Auto_time2.Pt:=real_to_time(3600);
    Run_Timer.En:=True;
```

```
   Auto_time1.En:=True;
   Auto_Tune_Running:=True;
   stable:=True;
   (*end_if;*)
end_if;

if ( Auto_Tune_Running )and (stable) then
   Auto_Tune_Start:=False;
   (* This while loop keeps the setpoint high unitl the target value is reached *)
   while ( Target_Sp-PIDControl.Pv < 0 and not (Auto_Tune_Stop) ) do
        PIDControl.Out:=( User_Target-Range );
        (*This If structure records the cycle time of the upper relay into Period1*)
        If ( Auto_time1.En ) then
           Auto_time1.En:=False;
           Auto_time2.En:=True;
           If (Time_to_real (Auto_time1.Et) <> Period1[1] ) then
              Period1[5]:=Period1[4];
              Period1[4]:=Period1[3];
              Period1[3]:=Period1[2];
              Period1[2]:=Period1[1];
              Period1[1]:=Time_to_real(Auto_time1.Et);
           end_if;
        end_if;
        (* This If Statement records the highest peak of this cycle*)
        If (PIDControl.Pv > Cycle_Hi) then
            Cycle_Hi:=PIDControl.Pv;
        end_if;
        Elapsed_Time:=(time_to_real(Run_Timer.Et))/60.0;
   end_while;
(*These 5 statements keep track of the five previous cycle peaks in Max_Pv and reinitilizes Cycle_Hi*)
If NOT ( Auto_Tune_Stop ) and NOT(Cycle_Hi = PIDControl.PvEuLo) then
   Max_Pv[5]:=Max_Pv[4];
   Max_Pv[4]:=Max_Pv[3];
   Max_Pv[3]:=Max_Pv[2];
   Max_Pv[2]:=Max_Pv[1];
   Max_Pv[1]:=Cycle_Hi;
   Cycle_Hi:=PIDControl.PvEuLo;
End_If;
   (*This is the lower half of the relay response cycle*)
   while ( Target_Sp-PIDControl.Pv >= 0 and not (Auto_Tune_Stop) ) do
        PIDControl.Out:=( User_Target+Range );
       (*This If structure records the cycle time of the lowr relay into Period2*)
       If ( Auto_time2.En ) then
           Auto_time2.En:=False;
           Auto_time1.En:=True;
           If (Time_to_real (Auto_time2.Et) <> Period2[1] ) then
              Period2[5]:=Period2[4];
              Period2[4]:=Period2[3];
              Period2[3]:=Period2[2];
              Period2[2]:=Period2[1];
              Period2[1]:=Time_to_real(Auto_time2.Et);
           end_if;
```

```
        end_if;
        (* This If Statement records the lowest peak of this cycle*)
        If ( PIDControl.Pv < Cycle_Low ) then
            Cycle_Low:=PIDControl.Pv;
        end_if;
        Elapsed_Time:=(time_to_real(Run_Timer.Et))/60.0;
    end_while;
(*These 5 statements keep track of the five previous cycle lows in Min_Pv and reinitilizes Cycle_Low*)
If NOT( Auto_Tune_Stop ) and NOT (Cycle_Low = PIDControl.PvEuHi) then
    Min_Pv[5]:=Min_Pv[4];
    Min_Pv[4]:=Min_Pv[3];
    Min_Pv[3]:=Min_Pv[2];
    Min_Pv[2]:=Min_Pv[1];
    Min_Pv[1]:=Cycle_Low;
    Cycle_Low:=PIDControl.PvEuHi;
End_if;
end_if;
(*Analysis of Cycle data to calculate Pu and Ku*)
count:=0;
Avg_Max :=0;
Avg_Min:=0;
Period:=0;
for count:= 1 to 5 do
   Avg_Max:= Avg_Max+Max_Pv[count];
   Avg_Min:= Avg_Min+Min_Pv[count];
   Period:= Period+Period1[count]+Period2[count];
end_for;
Avg_Max:=Avg_Max/5.0;
Avg_Min:=Avg_Min/5.0;
Amplitude:=(Avg_Max - Avg_Min)/2.0;
NormalizedAmplitude:=Amplitude/(PIDControl.PvEuHi-PIDControl.PvEuLo);
NormalizedRange:=Range/100.0;
Ku:=(4/3.14)*(NormalizedRange/NormalizedAmplitude);
Pu:=Period/5.0/60.0;
```