Low Cost Experimental Kits for Undergraduate Process Control Education

Christopher E. Long, Jeremy Wright, Charles E. Holland, and E. P. Gatzke

Department of Chemical Engineering

University of South Carolina, Columbia, SC 29208

Abstract

In process control education there is a need for hands-on experimentation. Students should see applications of theoretical coursework that is typically presented to them in the curriculum. Unfortunately, many commercially available experimental systems are prohibitively expensive. This work presents low cost experimental kits for use in undergraduate process control education. The prototype for a temperature control experiment using a standard household light bulb is detailed. Technologies for remote experimentation are also discussed.

Introduction

Undergraduate chemical process control courses typically present a fundamental basis of modern chemical process operation focusing on topics such as dynamic system modeling and traditional feedback control. The theoretical concepts are often difficult to convey to students, particularly without providing them with the opportunity to actually apply what they have been taught.

For this reason, a number of process control experiments have been developed and used across the country to reinforce the material covered in lectures [2–4, 6–9]. The hands on experiments provide the students with the opportunity to actively explore the connection between the theoretical content of the course and its application on real physical systems, often with the additional benefit of active learning in a small group environment [5, 11]. Unfortunately, many commercially available experimental systems are complex and thus too expensive, as not all institutions have the resources to provide each student with an adequate opportunity to experiment with such an apparatus. This work addresses this need through the development of a number of simple, extremely low cost experimental systems.

A series of inexpensive experimental kits have been designed for use in undergraduate process control education. Each simple system involves a number of analog and/or digital process inputs and outputs that are both observable and tangible thus lending to their pedagogical value. The kits utilize a USB interface data acquisition board to connect with software packages such as MATLAB/Simulink [10] and LabView [1]. This allows the students to propose and carry out a number of open- and closed-loop activities. One such system involves the use of a DC controlled dimmer to manipulate the intensity of a standard light bulb, which intern affects an adjacent temperature measurement taken using a thermocouple. In this case, students can easily collect open-loop data and model the relationship between the intensity of the light bulb and the surrounding temperature. Additionally, they can implement closed-loop control schemes to maintain the desired temperature at various setpoints in the presence of process disturbances such as cooling caused by the introduction of a fan. These portable kits offer a truly low cost means to incorporate hands on application of theoretical concepts into the typical process control course.

Traditional process control experiments are typically used in a lab or classroom environment. Time constraints can prevent students from completing their studies during normal class hours and the inconvenience of travel might prevent "distance learning" students from having the experience. Therefore, there is a seemingly great benefit in providing remote access to such experiments. In recent years this has been made possible via the internet. In this work, a readily available free open-source software package, Virtual Network Computing (VNC) is used for this purpose.

This work is organized as follows: a summary of the ideal experimental characteristics motivating this work is provided. The prototype of the low cost temperature control experiment is detailed. The uses of such an experimental kit are demonstrated. Finally, technologies for remote experimentation are discussed.

Experiments in Process Control

In setting out to design a number of experiments for process control education, a number of considerations must be made. First and foremost is safety. The experimental setup must be environmentally friendly with minimal risk to both the user and the equipment during operation. Ideally, the system would also be industrially relevant to help the students make the connection between classroom topics and engineering practice. The experiments need not be too complex as they could become intimidating and confusing. Nevertheless, they must not be so simple that they are taken for granted. Tangible and/or visible characteristic process variables, both inputs (manipulated variables) and outputs (measurements), lend to the overall pedagogical value. They allow the students to directly observe and understand the changes the system is undergoing in both open- and closed-loop operation. Reasonable process time constants are also important. Lengthy time constants could bore the students as they would have to constantly wait on the system to react. Alternatively, extremely small time constants might make process changes so quick that the student cannot absorb or comprehend what is going on. Each of these characteristics directly impacts the educational value of the experiment.

A number of vendors such as Feedback and Armfield have addressed these issues and offer commercially available experiments. However, the experiments are not perfect as they often do not lend themselves well to availability and convenience of use, which often dictate the viability of such a teaching approach. These systems can also be prohibitively expensive. For hands-on learning experiences to to become more commonplace in process control education, new experiments must provide a more cost effective means to demonstrate pertinent material. Lower cost setups obviously allow for academic departments or institutions to have access to more "units". This presumably leads to the use of many units by smaller groups or individuals as opposed to a larger group studying on a single expensive apparatus. Likewise, the lower cost can provide the institution the opportunity to invest in a variety of low-cost experiments to demonstrate different concepts or the same concepts in different contexts. An equally important issue is the sheer size of the system. Being compact and light weight promote portability such that these systems can be moved from the labs to the classroom, and in some cases taken home by students for further study. Ultimately, it would be desirable for each student to have sole access to a single portable experimental kit with inexpensive configurable parts that allow for a number of appropriate experimental studies.

One could imagine a number of experimental setups that possess many of these desirable attributes including being inexpensive and portable. These could include, but are not limited to the following: liquid-level control in a tank where a flow rate is manipulated to maintain a level in a continuously draining tank, concentration control in a mixer or reactor in which the flow rate of a number of liquid species is manipulated to maintain a specified product concentration, pressure control in which a gas flow rate is manipulated to maintain pressure in a constant volume tank, and temperature control in which the intensity of a light-bulb is manipulated to regulate the surrounding air temperature (presented below).

Temperature Control: A Light-bulb Experiment

A prototype of an inexpensive experimental kit for process control education has been fabricated. The system considered is a single-input single-output (SISO) system involving a standard household light-bulb. The experiment is pictured in Figures 1 and 3, as well as depicted schematically in Figure 2. It is designed such that the intensity of the light-bulb is changed to impact the temperature of the ambient air. A standard 12-volt PC case fan is placed adjacent to the light bulb to provide another dimension to the system. The fan can be discretely switched between a single speed and the off position. This can conveniently serve as a disturbance on the system during closed-loop operation.

The complete system consists of a limited number of rather inexpensive components. First and foremost is the data acquisition board. This board is an important piece of any experimental kit as it acts as the flexible backbone through which the other parts are integrated. Unfortunately, this piece can also prove to be the most expensive. In this study, two options were explored. The first option was the Velleman USB Experiment Interface Board (K8055). This board allows for up to five digital inputs and eight digital outputs, as well as two analog inputs and outputs. The board is easily interfaced to relevant software packages such as MATLAB/Simulink and LabView. Its power is supplied through a USB connection to an adjacent computer and the card is available for approximately forty dollars. However, the card only offers 8-bit resolution on the analog signals. Although this card was very inexepensive, it was found to be mostly insufficient for the process control system due to the low accuracy of the analog input.

In instances in which higher resolution is desirable and for this work, the more expensive (~$140) National Instruments Multi-function Data Acquisition for USB (NI USB-6008) can be used as it provides 12-bit resolution. This card allows for up to twelve digital input and output channels, as well as twelve analog inputs and two analog outputs. Again, this system is compatible

Component	Estimated Cost ($)
DC controlled dimmer	$20
12V case fan	$5
DAQ board w/ USB interface	$140
Standard Light bulb socket	$5
60 watt light bulb	$1
On/Off Switch for fan	$1
Temperature Sensor	$15
Total Cost:	< $200

Table 1: Complete Parts List with Cost Estimates

with MATLAB/Simulink and LabView. The interface is carried out using an MATLAB S-function written in C (Appendix). For this data acquisition system, the power is again supplied through the USB.

The voltage supplied to a DC dimmer is manipulated to operate the light bulb. In this case, a Velleman DC controlled dimmer is used. This unit was operated at 120 VAC. The max load on the dimmer is 3.5 A, while the control voltage range of this specific dimmer is between 0 and 10 VDC. This modified such that an input voltage range from 0-5 V scaled to a range of 0-120V supplied to the bulb. This dimmer acted as an adequate method for manipulating the light bulb intensity and is available for approximately twenty dollars. The temperature measurement is secured using a Lego temperature sensor from Mindstorms Robolab Technic RCX. This thermocouple is placed in close proximity to the light-bulb to measure the surrounding air temperature and is available for approximately fifteen dollars.

The estimated costs of all components used in the prototype is summarized in Table 1. From this it can be seen that the complete kit can be built for less than two hundred dollars. The cost of the data acquisition approach dominates the total cost. If a lower resolution output signal is acceptable, again there are lower cost alternatives. With the less expensive 8-bit Velleman board, the complete kit can be assembled for well less than one hundred dollars.

Demonstration of Uses

As mentioned before, the low cost nature and portability of the experimental kits allows for their use for classwork and homework alike. Effectively, the experimental kit is not a single assignment, but rather an experimental system that can act as a testbed for any number of studies in a variety of contexts throughout the semester of an undergraduate controls course.

Initially, students can study the relationship between the process input and output. Specifically, the gain of the system can be determined across the complete range of operation. The nonlinearity of this particular system can then be seen by the generation of the steady-state locus (Figure 4). Similarly, students can perform open-loop step tests (Figure 5) on the system and derive appropriate system models.

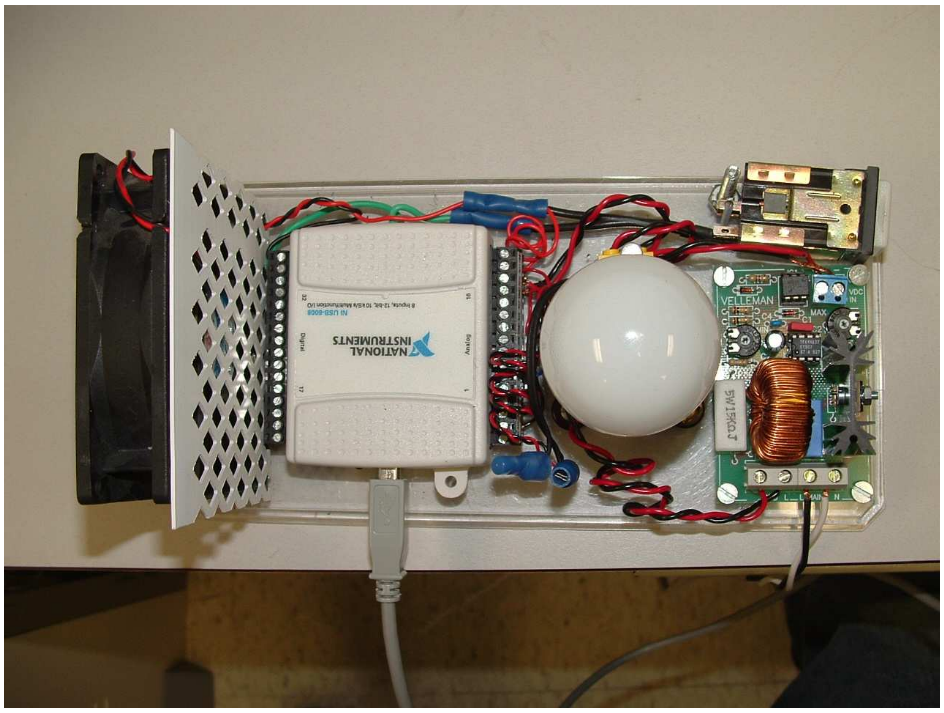

Figure 1: Photograph of the Low-cost Temperature Control Light-bulb Experiment (top view)

Students are able to develop and test control methodologies in MATLAB/Simulink such as Proportional-Integral-Derivative (PID) control and model-based control strategies such as Internal Model Control (IMC). Closed-loop results from a student developed PI-controller are presented in Figure 6. In this particular experiment, the students prescribed a step in the setpoint to move the desired target temperature from 80 to 100 degrees Fahrenheit 300 seconds into the run. At $t = 700$ seconds, a disturbance load is imposed on the system by manually switching the fan to the off position. The formulated controller is able to track the setpoint change and reject the disturbance appropriately.

Remote Experimentation

Hands-on experimentation typically supports the theoretical coursework covered in the traditional lecture periods. Students are able to spend some allotted time in a laboratory environment carrying out experiments. However, quite often the allotted time is not sufficiently long enough to allow each individual student to get an appreciable amount of time working alone on an experiment. Instead, time constraints typically force them into working in groups. Similarly, students having difficulty mastering certain material do not necessarily get an abundance of time. Students enrolled in “distance learning” programs are not always able to travel to participate in lab exercises. Thus, it is ideal to configure the experiments such that they can be accessed remotely via the internet. This way “distance learning” students can experience the application of the coursework and the full-time students can carry out their work around the clock at their convenience from

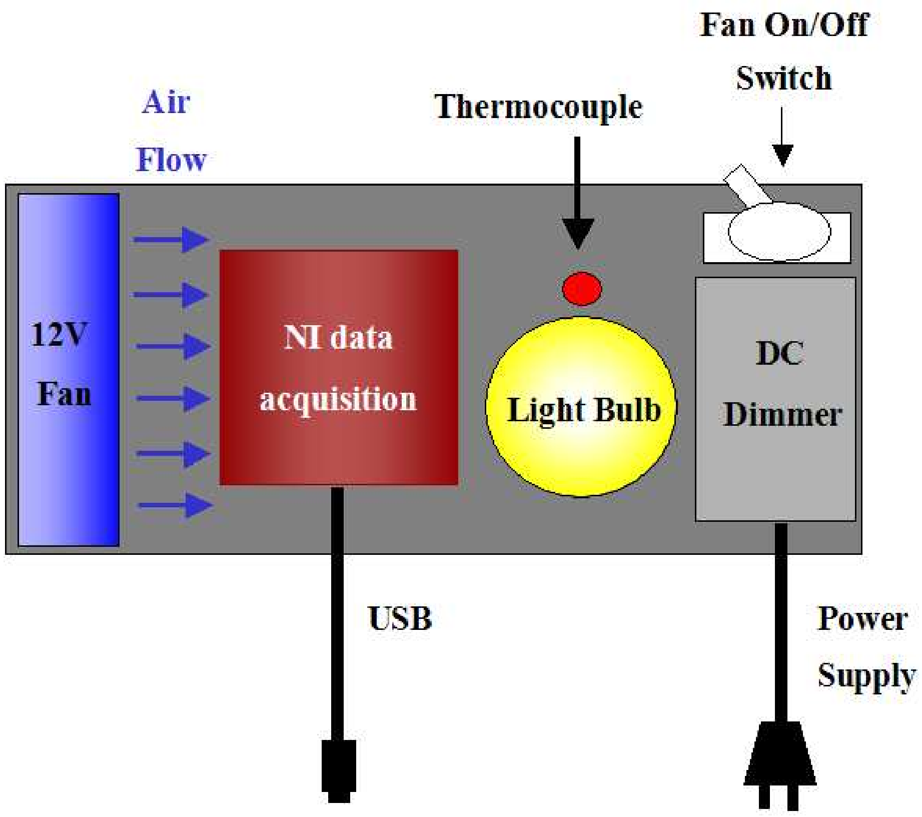

Figure 2: Schematic of the Low-cost Temperature Control Light-bulb Experiment (top view)

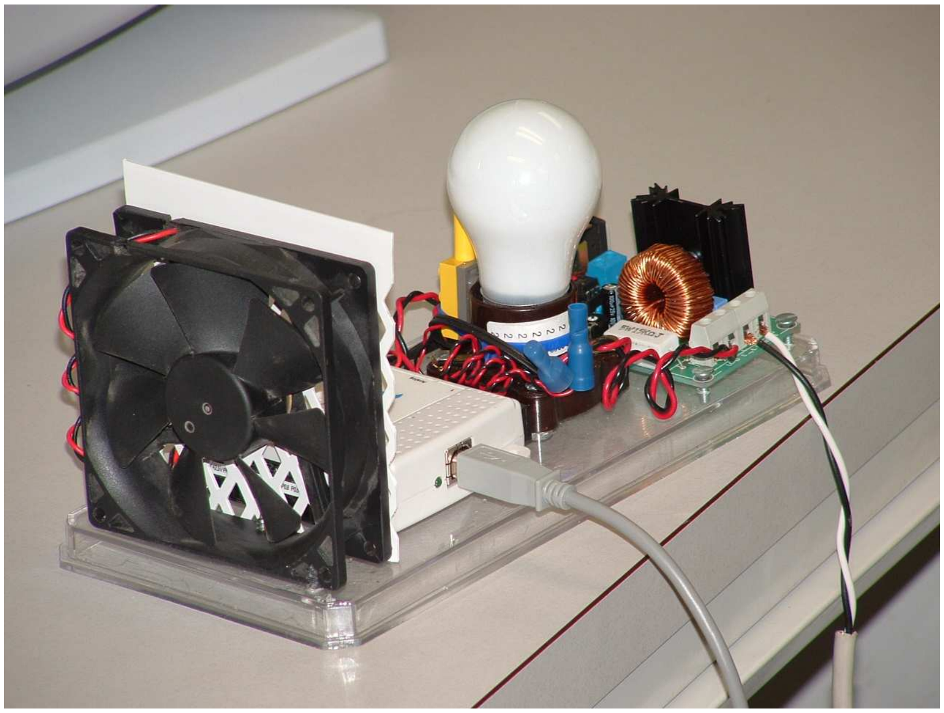

Figure 3: Photograph of the Low-cost Temperature Control Light-bulb Experiment (side view)

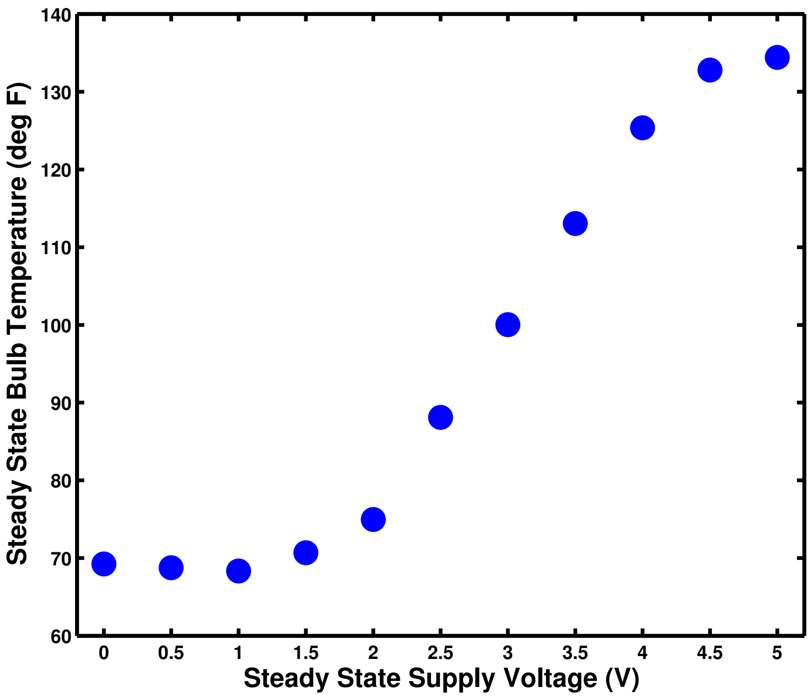

Figure 4: Steady-State Locus for the Temperature Control Light-bulb Experiment

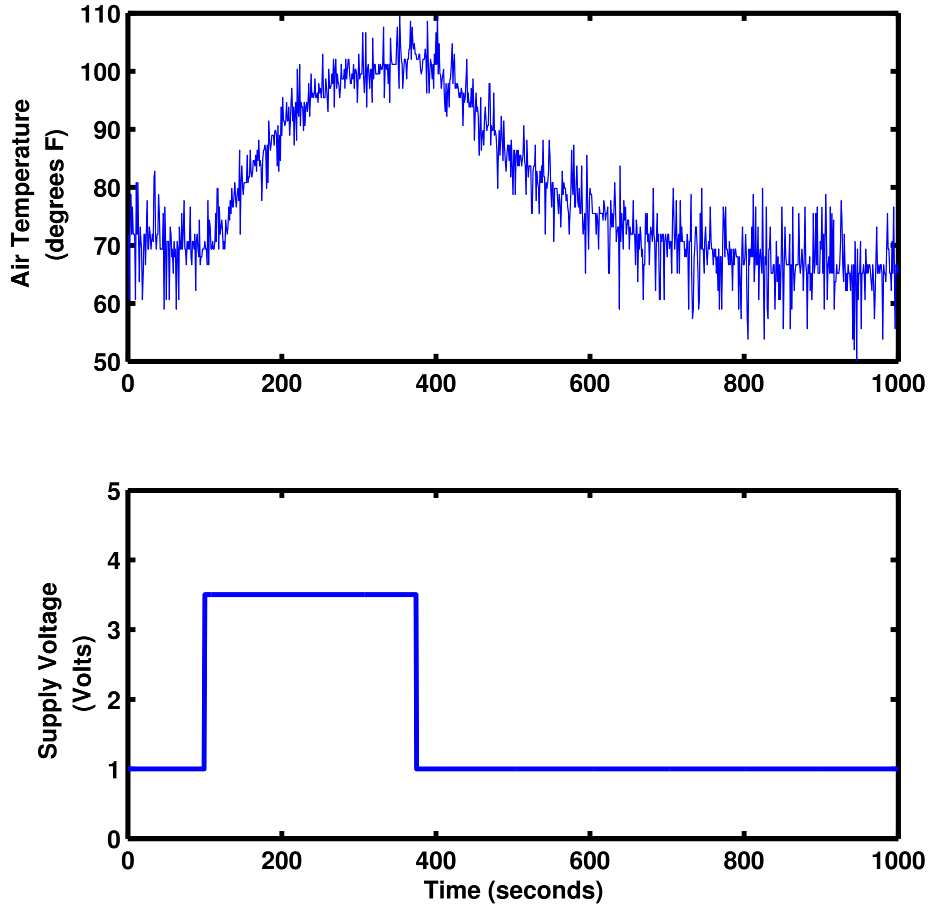

Figure 5: Open-loop Step test for Transfer Function Modeling

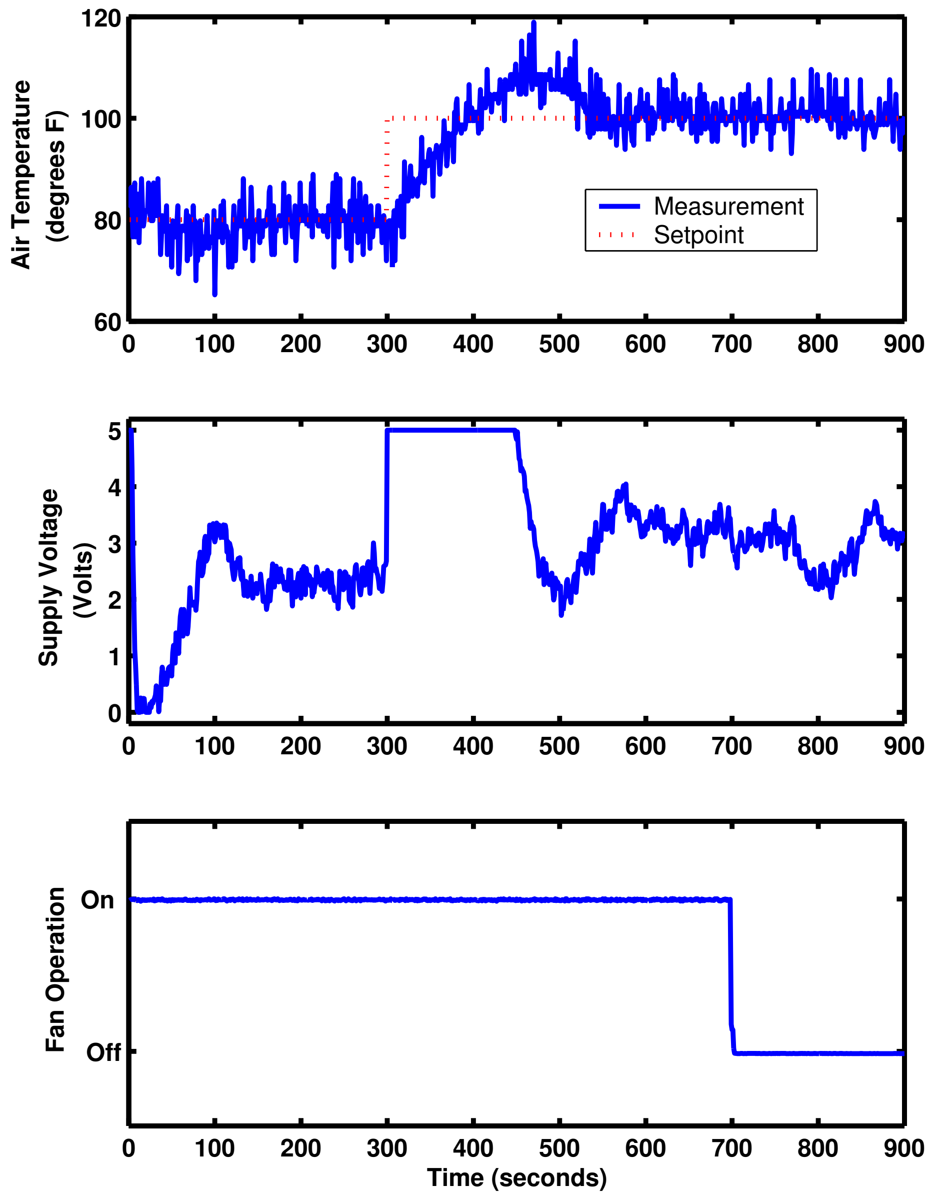

Figure 6: Closed-loop Results Depicting both Setpoint Tracking and Disturbance Rejection under Control of a PI controller Developed and Tuned by a Students.

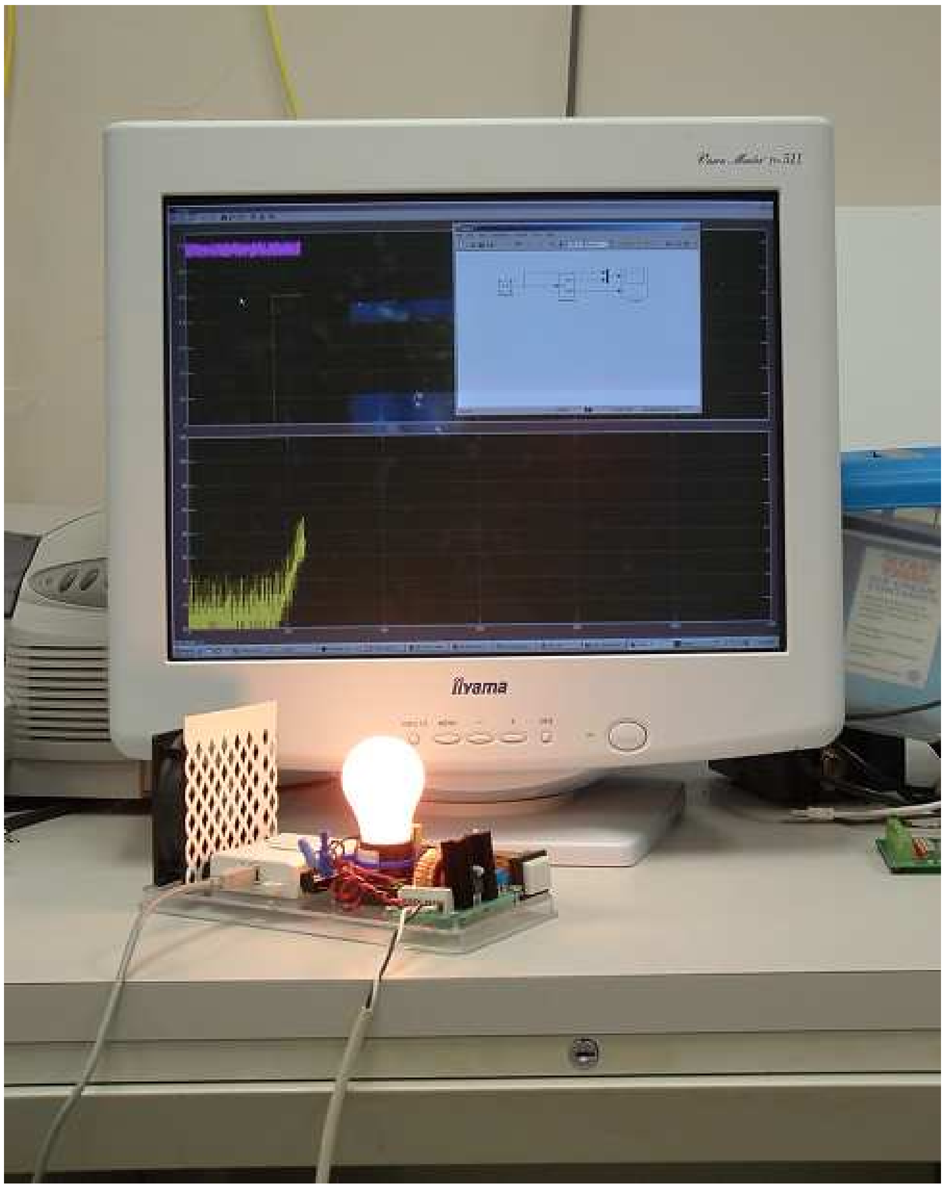

Figure 7: Photograph of the Temperature Control under Remote Operation while Attached to a Host Computer

computer labs, dorms, etc.

Remote access to the experiment was provided using Virtual Network Computing (VNC) software. This is an open-source package available for free online, in its most basic form, at *www.vnc.com*. The experiment can be attached to a host computer that is running a VNC server. Students then use a VNC viewer on their personal computer to access the host remotely and carry out experiments. In accessing the host computer, the remote operated effectively takes control of the host machine just as if he were sitting directly in front of it. That is to say that the display on the host machine is visible to the user.

The host computer is identified by IP address. The software is configurable to allow for password authentication to govern who has access to the equipment operation. Settings allow for or prevent multiple simultaneous (shared) connections to the equipment. This way remote users can work independently or in small groups with members in different locations depending on the nature of the assignment.

Conclusions

Hands-on experimentation is an important aspect of process control education. To this end, this work discussed the development of low cost experimental kits that allow this educational experience to be more readily available for everyone. In particular, a SISO temperature control experiment using a simple household light-bulb is detailed. Furthermore, technologies for remote access to the experiments that provide a convenient way for students to learn away from the structured lab/classroom setting is presented.

Acknowledgements

The authors would like to acknowledge financial support from the National Science Foundation Early Career Development grant CTS-038663.

References

[1] *LabVIEW 6.1*. National Instruments Corporation, 2003.

[2] Siong Ang and R. D. Braatz. Experimental Projects for the Process Control Laboratory. *Chemical Engineering Education*, 36(3):182–187, 2002.

[3] F. J. Doyle III, E. P. Gatzke, and R. S. Parker. "practical case studies for undergraduate process dynamics and conto using the process control modules. *Comp. App. in Eng. Edu.*, 6(3):181–191, 1998.

[4] F. J. Doyle III, E. P. Gatzke, and R. S. Parker. *Process Control Modules - A Software Laboratory for Control Design*. Prentice Hall, 1999.

[5] D. W. Johnson, R. T. Johnson, and K. A. Smith. *Active Learning: Cooperation in the College Classroom*. Interaction Book Co., Edina, MN, 1998.

[6] B. Joseph, C. Ying, and D. Srinivasagupta. Alaboratory to Supplement Courses in Process Control. Winter, 2002.

[7] J. H. Jung, M. Lee, J. Lee, and C. Han. A Development of Experimental Education Program: Computer Control of Multi-Stage Level Control System. 24(2):1497–1502, 2000.

[8] C. E. Long and E. P. Gatzke. Education application of an Experimental Four Tank System. submitted, 2005.

[9] T. E. Marlin. The Software Laboratory for Undergraduate Process Control Education. 20:S1371–S1352, 2000.

[10] The MathWorks. MATLAB® 6.5. Prentice Hall, 2002.

[11] B. J. Millis and P. G. Cottel. *Cooperative Learning for Higher Education Faculty*. Oryx Press, Phoenix, AZ, 1998.

Appendix

This appendix contains the source code for nidaq.c, the MATLAB S-function used for the interface.

```
#include "NIDAQmxBase.h"
#include <stdio.h>

#define S_FUNCTION_NAME nidaq
#define S_FUNCTION_LEVEL 2

#define NUM_INPUTS 1    /* Don't change this one */
#define INPUT_0_WIDTH 1 /* How many inputs to model */
#define INPUT_0_FEEDTHROUGH 1
#define NPARAMS 0
#define SAMPLE_TIME_0 0.1
#define NUM_DISC_STATES 1
#define DISC_STATES_IC [0]
#define NUM_CONT_STATES 0 /* Number of continuous states in the model */

#define NUM_OUTPUTS 1 /* Don't change */
#define OUTPUT_0_WIDTH NUM_CONT_STATES+2 /* how many things you want to output */

#define SFUNWIZ_GENERATE_TLC 1
#define SOURCEFILES ""
#define PANELINDEX 5
#define SFUNWIZ_REVISION 1.0

#include "simstruc.h"

void Outputs_wrapper(const real_T *u, real_T *y, const real_T *xC);
void Update_wrapper(const real_T *u, const real_T *y );
void Derivatives_wrapper(const real_T *u, const real_T *y, real_T *dx, real_T *xC);
```

```
// Analog input
TaskHandle taskHandlei = 0;
// Channel parameters
char chani[] = "Dev1/ai0,Dev1/ai1";
float64 mini = 0;
float64 maxi = 5;
// Timing parameters
uInt64 samplesPerChani = -1;
float64 data[2];
int32 pointsToRead = 1;
int32 pointsRead;
// Analog Output
TaskHandle taskHandleo = 0;
TaskHandle taskHandleo1 = 0;
// Channel parameters
char chano[] = "Dev1/ao0";
char chano1[] = "Dev1/ao1";
float64 mino = 0.0;
float64 maxo = 5.0;
// Timing parameters
uInt64 samplesPerChano = 1;
float64 power=5;
float64 power1=5;
int32 pointsWritten;
float64 timeout = 10.0;
// Digital Output
char chando[] = "Dev1/port0";
// Write parameters
uInt8 w_data[1];
int32 written;
static void mdlInitializeSizes(SimStruct *S)
{
    DECL_AND_INIT_DIMSINFO(inputDimsInfo);
    DECL_AND_INIT_DIMSINFO(outputDimsInfo);
    ssSetNumSFcnParams(S, NPARAMS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
       return;/* Parameter mismatch will be reported by Simulink */ }
ssSetNumContStates(S, NUM_CONT_STATES);
ssSetNumDiscStates(S, NUM_DISC_STATES);
if (!ssSetNumInputPorts(S, 1)) return;
ssSetInputPortWidth(S, 0, INPUT_0_WIDTH);
ssSetInputPortFrameData(S, 0, FRAME_INHERITED);
ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
ssSetInputPortRequiredContiguous(S, 0, 1); /*direct input signal access*/
if (!ssSetNumOutputPorts(S,1)) return;
ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
ssSetOutputPortFrameData(S, 0, FRAME_INHERITED);
ssSetNumSampleTimes(S, 1);
ssSetNumRWork(S, 0);
ssSetNumIWork(S, 0);
ssSetNumPWork(S, 0);
ssSetNumModes(S, 0);
ssSetNumNonsampledZCs(S, 0);
/* Take care when specifying exception free code - see sfuntmpl_doc.c */
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE | SS_OPTION_USE_TLC_WITH_ACCELERATOR);
// Task parameters
//Thermocouple Input Task
DAQmxBaseCreateTask("",&taskHandlei);
```

```
DAQmxBaseCreateAIVoltageChan(taskHandlei,chani,"",DAQmx_Val_Cfg_Default,mini,
                             maxi,DAQmx_Val_Volts,NULL);
DAQmxBaseStartTask(taskHandlei);

//Lightbulb Power Output Task DAQmxBaseCreateTask("",&taskHandleo);
DAQmxBaseCreateAOVoltageChan(taskHandleo,chano,"",mino,maxo,DAQmx_Val_Volts,NULL);
DAQmxBaseStartTask(taskHandleo);

//Thermocouple Power Output Task
DAQmxBaseCreateTask("",&taskHandleo1);
DAQmxBaseCreateAOVoltageChan(taskHandleo1,chano1,"",mino,maxo,DAQmx_Val_Volts,NULL);
DAQmxBaseStartTask(taskHandleo1);
}

/* Or this one */
/* Function: mdlInitializeSampleTimes ==========================================
*Abstract:
* Specifiy the sample time.
*/
static void mdlInitializeSampleTimes(SimStruct *S)
{
   ssSetSampleTime(S, 0, SAMPLE_TIME_0);
   ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_INITIALIZE_CONDITIONS
/* Function: mdlInitializeConditions ==========================================
* Abstract:
* Initialize the states */

static void mdlInitializeConditions(SimStruct *S) {
real_T *xC = ssGetContStates(S);
}

/* Function: mdlOutputs ========================================================
*
*/
static void mdlOutputs(SimStruct *S, int_T tid) { const real_T *u = (const real_T*)
ssGetInputPortSignal(S,0);
real_T *y = ssGetOutputPortRealSignal(S,0);
const real_T *xC = ssGetContStates(S);

/* Calculate the process outputs in the wrapper function below: */
Outputs_wrapper(u, y, xC); }
#undef MDL_UPDATE /* Change to
#define to use the function */
#if defined(MDL_UPDATE)

static void mdlUpdate(SimStruct *S, int_T tid) { const real_T *u = (const real_T *)
ssGetInputPortSignal(S,0);
real_T *xD = ssGetDiscStates(S);
const real_T *y = ssGetOutputPortSignal(S,0);

/* If you need to make updates to the process, do them in this wrapper function.
* If no updating is necessary, leave this function empty */
Update_wrapper(u, y);
}

#endif /* MDL_UPDATE */

#define MDL_DERIVATIVES /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
/* Function: mdlDerivatives ===================================================
* Abstract:
* In this function, you compute the S-function block’s derivatives.
* The derivatives are placed in the derivative vector, ssGetdX(S).
*/
static void mdlDerivatives(SimStruct *S) {
       const real_T *u = (const real_T *)ssGetInputPortSignal(S,0);
       real_T *dx = ssGetdX(S);
       real_T *xC = ssGetContStates(S);
       const real_T *y = ssGetOutputPortSignal(S,0);
       Derivatives_wrapper(u, y, dx, xC);
}
#endif /* MDL_DERIVATIVES */
static void mdlTerminate(SimStruct *S)
{
     DAQmxBaseStopTask(taskHandlei);
     DAQmxBaseClearTask(taskHandlei);

   power=0;
   DAQmxBaseWriteAnalogF64(taskHandleo,samplesPerChano,0,timeout,
       DAQmx_Val_GroupByChannel,&power,&pointsWritten,NULL);
   DAQmxBaseStopTask(taskHandleo);
   DAQmxBaseClearTask(taskHandleo);

   power1=0;
   DAQmxBaseWriteAnalogF64(taskHandleo1,samplesPerChano,0,timeout,
       DAQmx_Val_GroupByChannel,&power,&pointsWritten,NULL);
   DAQmxBaseStopTask(taskHandleo1);
   DAQmxBaseClearTask(taskHandleo1);
}
#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration function */
#endif
#include "tmwtypes.h"
/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
void Outputs_wrapper(const real_T *u, real_T *y, const real_T *xC) {
// Data read parameters
power = u[0];
DAQmxBaseWriteAnalogF64(taskHandleo,samplesPerChano,0,timeout,DAQmx_Val_GroupByChannel,
                       &power,&pointsWritten,NULL); power1 = 5;
DAQmxBaseWriteAnalogF64(taskHandleo1,samplesPerChano,0,timeout,DAQmx_Val_GroupByChannel,
                       &power1,&pointsWritten,NULL);
DAQmxBaseReadAnalogF64(taskHandlei,pointsToRead,timeout,DAQmx_Val_GroupByChannel,
                       &data,samplesPerChani,&pointsRead,NULL);

y[0]=data[1];
y[1]=data[0];

/* output equations go here ! */

/* Blank function unless you need updates */
void Update_wrapper(const real_T *u, const real_T *y ) { }
void Derivatives_wrapper(const real_T *u,const real_T *y,
                         real_T *dx,real_T *xC) {

/* Comments */
/* derivatives go here! */
/*
dx[0]=0;dx[1]=0;dx[2]=0;dx[3]=0;dx[4]=0;
*/
}
```