ColdSim - A Simulator for Teaching Process Control

A. F. Gilbert & W.A. Gilbert, Dept. of Chemical Engineering,
Lakehead University, Thunder Bay, Ontario, Canada, P7B 5E1

Abstract

ColdSim is a library of C/C++ routines to simulate and provide a graphic interface for the teaching of process control structure. It presents a process flowsheet on which controls can be configured and executed, using the block programming concepts of distributed control systems. Single loop, cascade, midranging, constraint and variable structuring controls, as well as feedforward controls can be configured. Versions of the library are available for SUN Solaris and Linux.

KEYWORDS: Process control, control structure, feedforward, cascade, simulation.

Introduction

Process control courses are rarely cited when chemical engineering graduates are surveyed on the university courses that best prepared them for their careers. More often than not, process control courses are a remembered as too theoretical and irrelevent to industrial practice. Much effort is spent on dynamic equations, transfer functions and stability analysis, culiminating in tuning rules for loops which instrumentation personnel tune quite satisfactorily, often by trial and error. Academics offer the defence that a proper understanding of process dynamics may be crucial to the more problematic cases which the graduate may face.

Another approach might be to include ideas on structure in control courses to balance the theoretical material. Chemical engineers are most often involved in the overall specification of control strategy, either for new processes or for refinements made to operating plants. Yet the only structural ideas they might remember from control engineering curriculum are about cascade and feedforward control. Moreover, the textbook treatment of feedforward control usually limited to transfer function models leading to the specification of lead/lag elements. Rarely is the student exposed to the block programming techniques of distributed control systems (DCS) or the absolute heat and material balance approach to feedforward control which is more natural to this environment.

Some resources on control structure exist. Shinskey's text on process control [1] is somewhat advanced for undergraduate course. His volume on multivariable control [2] gives a useful classification of plant controls as production, inventory, environmental, product quality and economic controls. It also presents many useful examples of advanced structures including cascade control, midranging, constraint controls, variable structuring and feedforward control. Other sources include Potvin [3].

One difficulty with teaching control structure is that students rarely have a developed sense of degrees of freedom or cause and effect in processes. When faced with a process flowsheet on which to draw simple feedback loops, it is amazing how often controlled variables are paired with manipulated variables that can have no possible impact, or how often conflicting control valves are placed on the same flow. Process common sense comes with experience, and this is difficult to provide in academia. The intention of the ColdSim process simulator discussed here, is to develop a sense of process through live, interactive, process flowsheets, rather than paper versions. The idea is to allow students to try out different control structures and to figure out for themselves what works and what does not.

Concept

ColdSim is a dynamic process simulator with a graphic user interface much like a process flowsheet or DCS console display. The process graphic includes sensors and control valves which may be operated manually, or linked by the addition of controls. The simulator is called ColdSim because the simulated process typically starts cold, with all flows off, and temperatures below that required for any chemical or physical process to take place. Tanks or other inventories may start empty, or they may contain initial levels of solvents and/or other components. Simulator calculations begin immediately when the program is started, but nothing happens until the user starts the feeds, turns on the steam, or does whatever else is required to get the process going. The process runs in real time, with trendplots to follow process performance.

ColdSim is available only of SUN Solaris 6+ or Linux, because it requires the gcc C/C++ compiler which is routinely available on these platforms. ColdSim is actually a library of routines [4] which provide a graphic interface based on OpenGL^{TM} [5]–[7], the GLUT windowing library [8], and the GLUI inteface library [9]. Trend plots were created from the plplot library [10]. The numerical integration is by the 4th order Runge-Kutta method [11].

To create a new simulation the user has only to prepare three things:

1. a process graphic (bmp file) drawn using any picture painting software.

2. a C/C++ user function with the derivative functions describing the process.

3. a model data file defining the initial state and independent variables of the simulation, and the location and size of instruments and valves on the bitmap.

The esthetics of the process graphic depend only on the artistic abilities of the designer. The model data file is easy to specify given the unambiguous initial conditions of a cold process. The most difficult task is to describe the process by derivative functions of state and input variables, based on heat and material balances. The combination of a cold start and real-time simulation makes debugging easier. Gross errors in heat and material balance modelling are found quickly at startup. Level controls can be configured before the feeds are introduced so as to avoid runaway integrating inventories. Energy inputs can be started, for example, by manually opening steam valves to see their effect, before implementing temperature controls. However, due to the time constraints in undergraduate curricula, the programming is a more suitable exercise for graduate course projects.

Once the model is compiled and run, simple feedback controls are added through a menuing operation which draws a movable PID controller icon on the graphic. Clicking on the icon opens a interface for the user to choose the controlled and manipulated variables. These links are also drawn on the process graphic. The loop can be made operational by entering setpoint and tuning parameters and by placing the controller in auto mode.

More advanced controls can be assembled using summers, multipliers, dividers, square root extractors, high and low selects, signal generators, and lead/lag compensators. Each of these elements is drawn as a movable icon on the process graphic, and their links are shown when connections are made via their respective interface windows. A complex graphic may look more like an instrumentation diagram than a process P&ID or a DCS operator interface, but the objective here is to teach control structure.

The control structure may be saved as a new model data file. The current values of all inputs and states are also saved so that the simulation can be restarted from this point. The new model data file is not fundamentally different than the starting one except for the new information about the controls.

Example CSTR Process

Figure 1 is an example process graphic of a continuous-stirred tank reactor (CSTR) process with controls. The image of the jacketed tank, lines and valves were created in a paint package. The sensor and valve vertical bar indicators are added dynamically by the ColdSim program, including the large sensor that doubles as the animated CSTR level. A PID controller links the level and the reactor discharge flow. Reactor

concentration control is cascaded to the reactor discharge temperature control manipulating the CSTR jacket cooling water.

The CSTR model is an exothermic reaction $A \longrightarrow B$ [12]:

$$r = k_0 e^{-E_a/RT_k} \quad (1)$$

$$\frac{dC}{dt} = \left(Q_f C_f - QC - VrC\right)/V \quad (2)$$

$$\frac{dT}{dt} = \left(Q_f \rho c_p T_f + Q_d \rho c_p T_d - Q\rho c_p T - \Delta H V r C - \Sigma_j (UA)_j (T - T_j)\right) / (V \rho c_p) \quad (3)$$

where V is the reactor volume (m^3), Q_f, C_f and T_f are the reactor total feed flow (m^3/s), concentration ($kmol/m^3$), and temperature (° C); Q_d, C_d, and T_d are the corresponding diluent variables; and Q, C and T are the respective discharge variables. The reaction rate r is by first order kinetics with Arrhenius rate constants k_0 (s^{-1}) and energy of activation E_a (kJ/mol), with T_k being the reactor temperature in °K. The heat of reaction is ΔH (kJ/mol). The density ρ (kg/m^3) and heat capacity c_p (kJ/kg-°C) are assumed to apply for all flows.

The reactor level model is simply:

$$\frac{dV}{dt} = \left(Q_f + Q_d - Q\right)/A_x \quad (4)$$

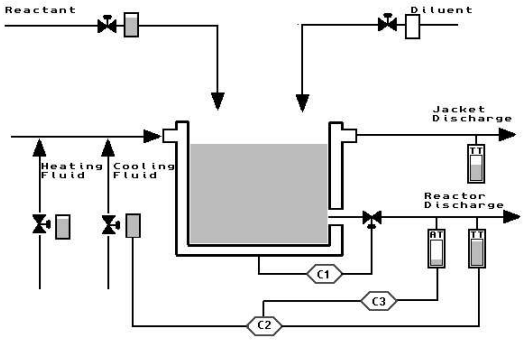
where Q_d is the diluent flow (m^3/s) and A_x is the cross sectional area of the tank (m^2).

The jacket is modeled as three series first order dynamic elements in heat transfer contact with the reactor contents with a single thermal capacity:

$$\frac{dT_j}{dt} = \left(Q_j c_j \rho_j (T_{j-1} - T_j) + \frac{(UA)_j}{3}(T - T_j)\right) / (V_j c_j \rho_j / 3) \quad (5)$$

where $(UA)_j$ is the heat transfer coefficient and area for the jacket sections $j = 1..3$; T_j, is the jacket section temperature; and Q_j, c_j, ρ_j are jacket flow, fluid heat capacity, and density, the same in all sections. The entering jacket flow and properties are determined by heat balance mixing of the cooling and heating fluid sources.

Figure 1: **CSTR Process Graphic**



The abbreviated C/C++ code is given in Table 1, and a partial model data file in Table 2. It is not difficult to create a simple model such as this. The only details not shown in the code of Table 1 are the preprocessor definitions of upper case variables like TANKC and DTANKC, which link these more easily read names to the model state derivative and state arrays dydt[] and y[], and statements to prevent inappropriate negative or zero values.

The model data file in Table 2 is entered as lines beginning with single character labels. A required first line is the model description 'd' specifying the number of integrator states, inputs and model constants, here 6, 4 and 14 respectively. The next line is model timing 't' described by a starting time, integration time interval (s) for the integration, the automatic exit time (s) and the output count. In this example, with a integration interval of 0.1 s and an output count of 10, calculation of control and refreshing of sensors, valves, amd trend plots, take place at 1 s intervals. Model states are specified by lines beginning with 'y' and an integer index, followed by the current value, minimum, maximum and the state name. Input variables are similarly defined by lines starting with 'x'. Normally the current state values are started with "cold" values, but the values in Table 2 are from a saved model.

The model constants are for variables that can be changed by the user during a simulation. A window interface containing these values may be opened by a menu operation. This model has 14 such constants, some of which are shown in Table 2. Not all program constants need be treated this way. Most can be hardwired in the program code.

Sensors and valves are described by lines beginning in 's' and 'v' respectively and an integer identifier. The integer identifier corresponds to

the state variables (y) for sensors, or to the inputs variables (x) for valves. The sensor lines specify the percent noise, origin x, origin y, width and height in percent of the sensor indicator, on the graphic. The valve lines specify the linked controller (or -1, if none), the minimum and maximum and current percent opening, and origin x, origin y, width and height of the valve indicator. Regretably, positioning and sizing of the sensor and valve indicators is an annoying trial by error process.

The remaining lines beginning in 'f' describe the PID controllers, the details of which will not be explained here. Other control objects have their own descriptions. None need be specified to start a simulation. It is better to use the graphic interface to configure the controls, and then save the complete model.

Cascade Control

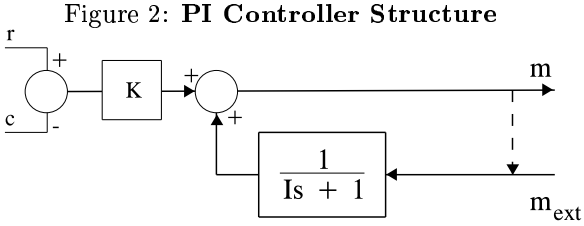
As a finer point in control structure, the PID controller algorithm permits the use of external feedback in the integral mode. External feedback is often essential for cascade control or for advanced feedback structures using high or low selects [2]. The algorithm [1] computes the integral contribution by a lag as shown in Figure 2. Normal internal feedback, shown by the dashed line causes the controller output m to wind up to 100%, or down to 0% (assuming these limits), whenever the error $e = r - c$ is not zero. On the other hand, if external feedback is used, the controller output will tend to follow m_{ext}, with a small proportional offset.

Thus ColdSim can be used to illustrate the impact of reset wind-up and what might be done about it. In cascade control, the requirement is to protect the primary loop against saturation in the secondary loop. The usual external feedback is the secondary loop measurement, which is most representative of the secondary loop setpoint, the primary controller's target.

In the cascade structure of Figure 1, the reactor primary concentration control sets the secondary temperature setpoint, which in turn manipulates the cooling water flow. In the trend plot of Figure 3, representing normal internal feedback, the secondary temperature control loop is near saturation, because of a limitation in cooling capacity. (Someone has left the heating fluid on!) Just after 200 s the cooling capacity is exceeded, causing a loss of control, as indicated by the high temperature (lower figure) and the low discharge concentration (upper figure). In response, the concentration controller's integral mode ramps down the temperature setpoint. Given enough time the temperature setpoint would be driven down to zero, if the constraint in the cooling system is not relieved. In this example, constraint is released at around 350 s, but because the temperature setpoint has wound down so far, it takes a long time for the reactor temperature and concentration to stabilize. Morover, a large postive overshoot occurs in the reactor concentration.

Figure 4 shows a similar sequence of events with the reactor temperature selected as external feedback to the concentration controller. When the cooling constraint occurs, the reactor temperature goes high and the reactor concentration goes low, as before. This is inevitable given the constraint. The important issue here is the behaviour of the primary controller during the loss of control. By feeding back the high, but more constant reactor temperature, the concentration controller output and temperature setpoint no longer wind down, but remains close the value before control was lost. In this way the control structure is better able to cope with the resumption of normal operating conditions, as shown after 360 s.

Figure 2: **PI Controller Structure**



Other Capabilities

In addition to single loop and cascade control, ColdSim is capable of demonstrating midranging, parallel positioning, sequenced valves, and the use of selective controls. The high and low selects allow for auctioneering, variable structuring, and constraint controls as illustrated in [2]. Absolute

style feedforward controls can be configured by block programming with square root, summers, multipliers, dividers and lead/lag blocks.

An example of midranging control is illustrated in Figure 5. The discharge concentration from the CSTR is controlled directly by the coolant flow. A valve position controller takes the temperature controller output as its measurement and maintains the coolant flow at 50% by manipulating the diluent flow. This may be a rather unusual use of midranging, but serves to explain the concept of a valve position controller.

At the present, the CSTR example is the only one ready for release with the libraries. ColdSim is capable of much larger models. There are no limits in the number of states or inputs allowed, but practical limits exist in terms of display, and computing requirements. It is hoped that others will take up the challenge to prepare other models, and to share them [4].

Figure 3: **Cascade with Internal Feedback**

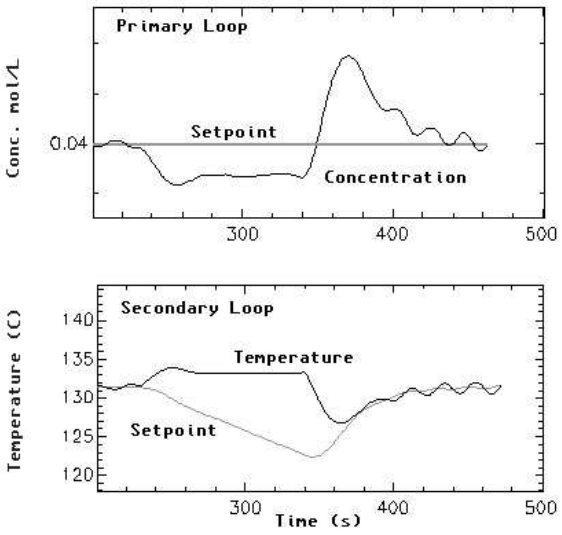
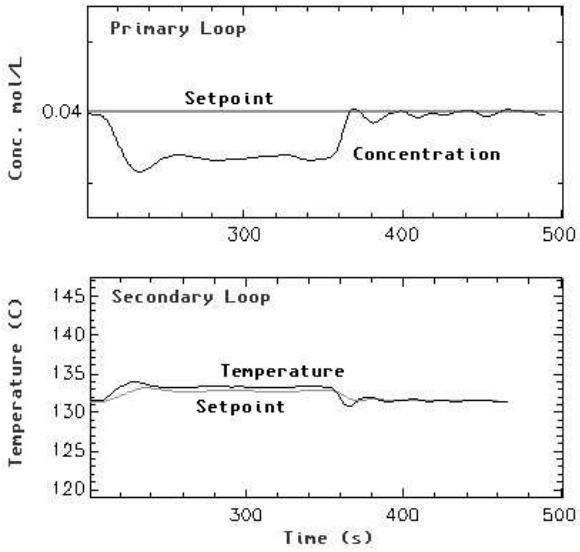


Figure 4: **Cascade with External Feedback**



References

[1] Shinskey, F.G. "Process Control Systems: Application Design and Tuning", 4th Ed., McGraw-Hill, New York (1996).

[2] Shinskey, F.G. "Controlling Multivariable Processes", Instrument Society of America, Research Triangle Park, North Carolina (1981).

[3] Potvin, J., "Applied Process Control Instrumentation", Reston Publishing Co., Reston, Virginia (1985).

[4] http://flash.lakeheadu.ca/~agilbert /ColdSim.html or allan.gilbert@lakeheadu.ca

[5] Wright, R.S. & Sweet, M., "OpenGL SuperBible", The Waite Group, Corte Madera, CA (1996).

[6] Kilgard, M.J., "OpenGL Programming for the X Window System", Addison-Wesley, Reading, MA (1996)

[7] http://www.opengl.org

[8] http://www.opengl.org/resources/libraries /glut.html

[9] http://www.nigels.com/glt/glui

[10] http://www.plplot.sourceforge.net/

[11] Press, W.H., Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P., "Numerical Recipes in C", 2nd Ed., Cambridge University Press (1992).

[12] Marlin, T.E., "Process Control - Designing Processes and Control Systems for Dynamic Performance", 2nd ed., McGraw Hill (2000).

Figure 5: **Midranging Process Graphic**

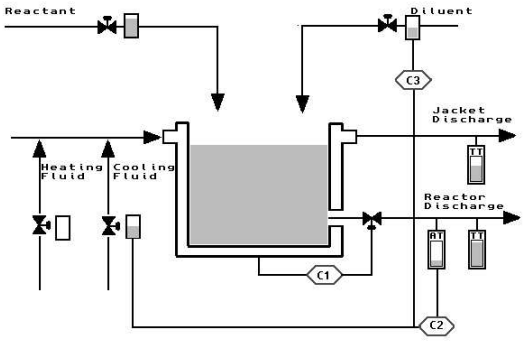


Table 2 Data File

```
d 6 5 14
t 0.0 0.10 7200.0 10
p 587 487 cstr.bmp
y0 0.0399 0.00 1.000 Reactor_Conc.
y1 131.46 0.00 150.0 Reactor_Temperature
y2 1.79    0.00 2.00   Reactor_Level
y3 95.27   0.00 200.0 Coolant_Temp_1
y4 98.39   0.00 200.0 Coolant_Temp_2
y5 101.2   0.00 200.0 Coolant_Temp_3
x0 0.032   0.00 0.040 Feed_Flow_(m3^/s)
x1 0.000   0.00 0.040 Diluent_Flow_(m3^/s)
x2 0.326   0.00 0.333 Coolant_Flow_(m3^/s)
x3 0.262   0.00 0.333 Heating_Fluid_Flow
x4 0.032   0.00 0.080 Reactor_Discharge
k0 166700000. Arrhenius_Const.
k1 8330.0 Arrhenius_E/R
⋮
k13 2.00 Jacket_Volume
s0 0.00 79.80 23.00 12.00 30.00
s1 0.00 86.80 23.00 12.00 30.00
s2 0.50 38.09 28.50 140.1 164.3
s5 0.00 86.69 46.50 12.00 30.00
v0 -1 0 100 80.0 27.00 87.00 11.00 24.35
v1 -1 0 100 0.00 75.99 87.00 11.00 24.35
v2 13 0 100 98.1 27.00 31.00 11.00 24.35
v3 -1 0 100 78.8 15.00 31.00 11.00 24.35
v4 12 0 100 40.0 68.69 26.00 11.00 24.35
f12 2 4 -1 -1 -6.0 0.10 0.00 A 1.80 40.0 57.58 82.13
f13 1 2 14 -1 -6.0 0.10 0.00 RP 131.45 98.14 50.42 95.27
f14 0 13 -1 1 -1.0 0.10 0.00 A 0.04 87.6 65.5 89.32
```

Table 1 Function of Derivatives

```
void f(float t,float *y, float *dydt )
{
float *x,*k,*ss,ert,ua,rk,uat,vtank,ht,htt,jhin,jackf;
int i,j;
vtank = ATANK*TANKL;
if (vtank <= 0.0 ) vtank = 0.01;
ert = ARRHER/(TANKT+273);
rk = ARRHK0*exp(-ert);
jackf = CFLOW + HFLOW;
ua = UAa*pow((double)jackf,(double)UAb);
if (ua < 50.000 ) ua = 50.000;
DTANKL = (FFLOW+DFLOW-RFLOW)
        /ATANK;
  /*heat to 1st jacket zone*/
jhin = RHOCPC*(CFLOW*COOLT
        +HFLOW*HEATT);
htt = 0;
  /*divide jacket in 3 mixing volumes*/
for (i=3; i< 6; i++) {
        /*zone heat transfer*/
  ht = ua/3 *(TANKT-JACKT[i]);
  DJACKT[i] = (jhin + ht -(CFLOW+HFLOW)
        *RHOCPC*JACKT[i])
        /(JACKV/3*RHOCPC);
  jhin = (CFLOW+HFLOW)*RHOCPC
        *JACKT[i];
  htt+=ht; /*total heat transfer*/
  }
DTANKC = 1/vtank*(FEEDC*FFLOW-
        TANKC*RFLOW) - rk*TANKC;
DTANKC -= TANKC*DTANKL/TANKL;
DTANKT = 1/vtank*(FFLOW*FEEDT
        +DFLOW*DILT-RFLOW*TANKT);
DTANKT -= htt/(vtank*RHOCP);
DTANKT += -HREACT*rk*TANKC/RHOCP;
DTANKT -= TANKT*DTANKL/TANKL;
return;
} /* end function f */
```