Object-oriented modelling of industrial PID controllers

Alberto Leva[a], Marco Bonvini[b], Martina Maggio[b]

[a]*Politecnico di Milano, Dipartimento di Elettronica e Informazione*
Piazza Leonardo Da Vinci, 32 - 20133 Milano, Italy
[a]*PhD student at the Dipartimento di Elettronica e Informazione*
{*leva,bonvini,maggio*}*@elet.polimi.it*

Abstract: This paper presents a library of (PID) controller models adopting the object-oriented approach, and written in Modelica. Peculiar to this work is that controllers are represented both as dynamic continuous-time and digital models, achieving consistence between the two and accounting for the functionalities of typical industrial implementations. This allows the designer to use realistic controllers, maintaining the possibility of choosing the continuous-time or digital (event based) representation. The former allows for example for variable-step simulation, to the advantage of efficiency, while the latter represents very realistically the actual control system's operation, clearly at the cost of more simulation time. Beside standard PI and PID controls, in this work also autotuning is (initially) considered, and some application examples are reported to show how the presented library can ease system studies involving (PID) controls.

Keywords: Object-oriented modelling and simulation, PID control, continuous-time control, digital control.

1. INTRODUCTION

Object-oriented modelling and simulation is nowadays a major tool to assess the behaviour of complex controlled systems. The reasons for that are numerous and impossible to discuss here; the interested reader can refer to Mattsson et al. (1998), Casella and Leva (2006), and many other works. Object-oriented languages and tools are widely used to assist engineering and control synthesis in a number of domains, ranging from process to automotive applications and more. In this work we refer to the Modelica language, however it is worth saying that the mentioned concepts are completely general and could in principle apply to any other object-oriented language.

Quite intuitively, in a large number of cases the object to be simulated contains some control, most frequently containing PID blocks. Curiously enough, however, at least in the authors' opinion and to the best of their knowledge, most of the available libraries for the simulation of controllers – in the addressed context of overall object-oriented control system models – do not focus on some relevant aspects of *industrial* controllers, thereby posing to the designer some nontrivial issues, that can be briefly summarised by looking at two extreme cases.

If the simulation study aims at devising a control strategy, the most natural way to go when constructing controller models is to adopt an equation-based approach in the continuous time domain, relegating events to really event-based control parts, so as to allow variable-step solvers to unleash their power in a view to maximise simulation efficiency. If this is the case, the standard blocks offered by the Modelica Standard Library, or some extensions like the `LinearSystems` library, are perfectly adequate. However, the resulting control descriptions will be adequate too for the analyst, but definitely too high-level for the people who need to turn them into functional specifications suitable for coding.

The opposite case is when one has to model an already existing controlled plant. In such a case, control-related information is normally provided in the form of programming diagrams (e.g., in an IEC61131.3-compliant language) from which the extraction of Modelica schemes is often not easy at all. One issue is certainly that programming diagrams normally contain a number of signals that are totally ancillary for any system-level simulation study and thus just need omitting, but there is another more relevant problem, as some core blocks of most controls (think for example to two controllers switched in and out alternatively with the inactive one tracking the active one) simply do not have a representation in the typical Modelica libraries. In such a case, ensuring that the controller in the model "is the same" as the controller in the plant may sometimes be tricky indeed.

A third case can be imagined as a combination of the two, if for example one thinks of a control strategy specified as a model, and an available realisation of it that needs checking for correctness against the model. Many other combinations of the above cases or of similar ones can be figured out, and the conclusion is that there must be some modelling aid "in between", the *extrema* being continuous-time models on one side, and controller code on the other.

Projecting this *scenario* on object-oriented modelling tools, and starting specialising to Modelica, one immediately observes a still incomplete exploitation, as far as the research addressed herein is concerned, of a very relevant peculiarity of the object-oriented modelling paradigm, namely the possibility of mixing equation- and algorithm-based modelling. Such a peculiarity allows for very detailed control system representations, in principle up to a full code *replica*, and if correctly exploited in

coordination with equation-based modelling, can provide an efficient solution to the issues sketched out above.

Following this idea, this paper presents a model library where controllers are represented *both* as continuous-time models, for efficient simulation, and as full-fledged industrial algorithmic implementation, to realistically assess the system operation: consistency is ensured between any couple of representations for the same object, concerning both interfaces and internal behaviour (apart from the differences inherently introduced by the digital realisation of course), so as to allow for easy replacement of all or part of the control system with one or the other representation, thereby tailoring simulation accuracy and efficiency in principle to any specific study. The presented library is meant to be the *nucleus* for a larger one.

At present the main focus is on standard PI and PID controls – that significantly contribute to form the backbone of industrial controls Dorf and Bishop (1995); Åström and Hägglund (2006) –, although autotuning is initially considered. The idea is to have a set of representations for each block, in principle down to individual implementations of it in a particular control engineering and/or configuration and programming tool.

The paper is organised as follows. Section 2 illustrates the library structure, while section 3 deals with the main issue of the work – i.e., having *consistent* continuous-time and digital models – and shows, with reference to a simple PI controller, that things need sometimes more attention than one may expect. Section 4 briefly describes some application examples, to show how the presented library can ease system studies involving (PID) controls, and section 5 deals with possible extensions. Finally, in section 6 some conclusions are drawn and future research is sketched out.

2. THE LIBRARY STRUCTURE

The structure of the `ControlLibrary` Modelica package is illustrated in figure 1.

For brevity only the "Continuous" subtree is expanded, thus hiding the "Discrete" counterpart of its component, and the "Logical" part is shown to illustrate how the library is designed in a view to represent realistic schemes. As anticipated the present version of the library is just something more than a *nucleus*, nonetheless it is already usable.

The library will very soon be released as free software within the terms of the Modelica license, at the URL `modelica.org`.

3. REPRESENTATION CONSISTENCY

3.1 Continuous and discrete time as interchangeable

The antiwindup PI controller with bias implemented in the library is here shown as an example of interchangeable continuous- and discrete-time blocks.

Both representations of the controller share the input/output structure and the parameter set. The controller is implemented in the continuous time domain with the block diagram of figure 2, which corresponds to the Modelica code of listing 1.

Note that in the library a more "industry-like" notation was deliberately used: the set point (y° in typical control textbooks) is termed `SP`, the controlled (or "process") variable (y) is denoted by `PV`, and the control signal (u) by `CS`.

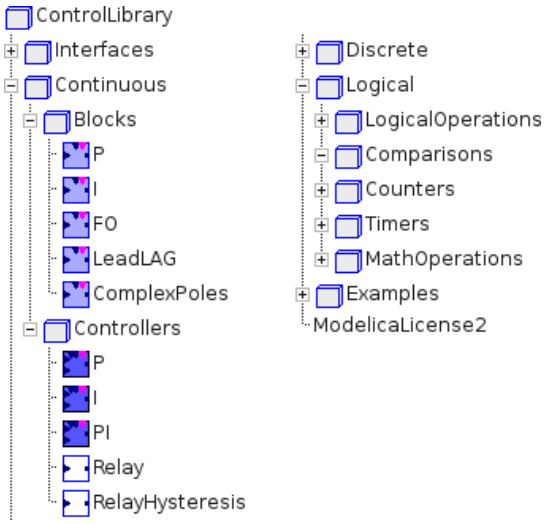

Fig. 1. Structure of the `ControlLibrary` Modelica package.

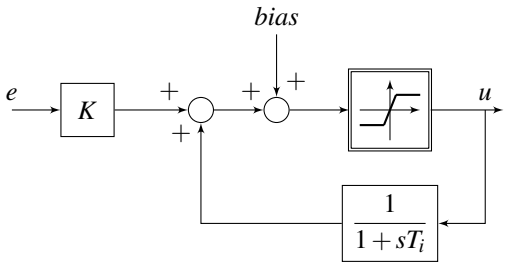

Fig. 2. Block diagram of the antiwindup PI controller.

Listing 1: continuous-time PI.

```
model PI
  "Continuous-time Proportional+Integral controller
   with AntiWindup, Tracking mode and Bias input"
  extends ControlLibrary.Interfaces.Controller;
  parameter Real K = 1
    "Proportional gain";
  parameter Real Ti = 1
    "Integral time";
  parameter Real CS_start = 0
    "Initial value of the control signal";
  parameter Real CSmin = 0
    "minimum value of the control signal";
  parameter Real CSmax = 1
    "maximum value of the control signal";
  parameter Boolean AntiWindup = true
    "Antiwindup presence flag ";
protected
  parameter Real eps = Ti/1e3;
    "Small time constant for auto/track switching";
  Real FBout
    "output (and state) of block 1/(1+sTi)";
  Real satin
    "input of the saturation block";
equation
  satin = FBout+K*(SP-PV) + bias;
  CS    = if AntiWindup
             then max(CSmin,
                  min(CSmax,(if ts then tr
                                else satin)))
          else satin;
  FBout = - Ti*der(FBout) + CS
          - (if AntiWindup then max(CSmin,
                                    min(CSmax,bias))
                           else bias);
 initial equation
   FBout = if AntiWindup then max(CSmin,
                                  min(CSmax,CS_start))
                         else CS_start
           - bias - K*(SP-PV);
end PI;
```

The same controller implemented in the discrete time, conversely, corresponds to the Modelica code of listing 2.

Listing 2: digital (event-based) PI.

```
 model PI
   "Digital Proportional+Integral controller
    with AntiWindup, Tracking mode and Bias input"
   extends ControlLibrary.Interfaces.Controller;
   parameter Real K = 1
     "Proportional gain";
   parameter Real Ti = 1
     "Integral time";
   parameter Real CS_start = 0
     "Initial value of the control signal";
   parameter Real CSmin = 0
     "minimum value of the control signal";
   parameter Real CSmax = 1
     "maximum value of the control signal";
   parameter Boolean AntiWindup = true
     "Antiwindup presence flag ";
protected
  discrete Real satin;
  discrete Real FBout;
  discrete Real cs;
algorithm
  when sample(0,Ts) then
     satin := pre(FBout)+K*(SP-PV)+bias;
     cs    := if AntiWindup
                then max(CSmin,min(CSmax,satin))
                else if ts then tr else satin;
     FBout := (Ti*pre(FBout)+Ts*cs)/(Ti+Ts);
  end when;
equation
  CS      = cs;
initial equation
  pre(FBout) = if AntiWindup
                 then max(CSmin,
                      min(CSmax,CS_start))
                 else CS_start
               - bias - K*(SP-PV);
 end PI;
```

3.2 Continuous and discrete time co-existing

The basic principle of relay-based autotuning was introduced in Åström and Hägglund (1984), and then developed in Åström and Hägglund (1991); Leva (1993); Besançon-Voda and Roux-Buisson (1997); Luyben (2001); Leva (2005) and many other papers. In a nutshell, the idea is to force the controlled variable to enter a permanent oscillation condition via relay feedback, employ said oscillation's characteristics to estimate one point of the process frequency response, and finally compute the regulator parameters so that one point of the open-loop frequency response be suitably assigned; a survey on the matter, for the interested reader, can be found in Yu (1999).

The autotuning PI block is used to show continuous- and discrete-time models of the same controllers that co-exist, the simulation switching from one representation to the other. The reason to do so is that autotuners are typically expressed as procedures and hardly realisable as dynamic system only, if not at the price of much complexity. It is then advisable to model an autotuner as a digital block only.

On the other hand, when no autotuning is in progress, there is no reason why the digital nature of the block should hamper efficiency by preventing e.g. the use of variable step solvers.

The way to go is then to have *both* descriptions of the block (the tuned regulator) in place, and activate the digital one only when autotuning is in progress. This is illustrated by the PI autotuner of listing 3.

The tuning procedure is relay-based, using the scheme of figure 3: the process frequency response point with phase -90° is found with relay plus integrator feedback, and then the PI is

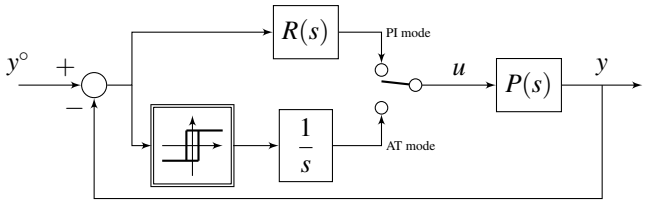

Fig. 3. Basic scheme for relay-based (PI) autotuning.

tuned for a specified phase margin. Since the used autotuning method is very well known, no further details on that are given.

Listing 3: autotuning PI.

```
model ATPIrelayNCmixedMode
  // ... declarations omitted for brevity ...
equation
  // Continuous-time antiwindup PI
  satIn = K*(SP-PV)+linFBout;
  CSpi  = Ti*der(linFBout)+linFBout;
  CSpi  = noEvent(max(CSmin,min(CSmax,satIn)));
  // Output selection
  if iMode==0 or iMode==1 then // 0, PI or 1, AT init
     CS = CSpi;
  else                        // 2, AT run
     CS = CSat;
  end if;
algorithm
  // Autotuning procedure
  when initial() then
       K  := K0;
       Ti := Ti0;
       AT := false;
  end when;
  // Turn on AT when required
  when ATreq and sample(0,Ts) then
       if not AT then
          AT    := true; // set AT flag on
          iMode := 1;    // set next mode to AT init
       end if;
  end when;
  // AT init mode, set next mode to AT run
  when AT and iMode==1 and sample(0,Ts) then
       CSat         := pre(CSpi);
       UP           := false;
       period       := 0;
       wox          := 0;
       Pox          := 0;
       rPVmax       := pre(PV);
       rPVmin       := pre(PV);
       rCSmax       := CSat;
       rCSmin       := CSat;
       lastToggleUp := time;
       nOx          := 0;
  end when;
  // AT shutdown;
  when (iMode==1 or iMode==2) and not AT
       and sample(0,Ts) then
       // re-initialise the continuos-time PI
       iMode := 0;
       reinit(linFBout,CSat);
  end when;
  // AT run mode
  when AT and iMode==2 and sample(0,Ts) then
        // Manage relay
        if UP==false and PV<=SP then
           UP    := true;
        end if;
        if UP==true and PV>SP then
           UP    := false;
        end if;
        if UP==true then
           CSat := CSat + slope*Ts;
        else
           CSat := CSat - slope*Ts;
        end if;
        // record relay id max and min for PV and CS
        if PV>rPVmax then
           rPVmax := PV;
        end if;
        if PV<rPVmin then
           rPVmin := PV;
        end if;
        if CSat>rCSmax then
           rCSmax := CSat;
        end if;
        if CSat<rCSmin then
           rCSmin := CSat;
```

```
        end if;
        // tune if perm ox
        if UP==true and pre(UP)==false then
           period       := time-lastToggleUp;
           lastToggleUp := time;
           if period>0 and nOx>=nOxMin
              and abs(period-pre(period))/period
                  < permOxPeriodPerc/100 then
              AT   := false;
              wox  := 2*pi/period;
              Pox  := pi^2*(rPVmax-rPVmin)/8
                      /(rCSmax-rCSmin);
              Ti   := tan(pm/180*pi)/wox;
              K    := tan(pm/180*pi)/(Pox
                      *sqrt(1+(tan(pm/180*pi))^2));
           end if;
           rPVmax  := PV;
           rPVmin  := PV;
           rCSmax  := CSat;
           rCSmin  := CSat;
           nOx     := nOx+1;
        end if;
  end when;
end ATPIrelayNCmixedMode;
```

For reference, the library also contains a fully digital version of the same autotuner, where no continuous-time PI is used. The code is omitted for brevity.

3.3 Some overall remarks

As can be seen, when turning the continuous-time controller into the discrete-time one, not only its linear behaviour but also nonlinear features like antiwindup need considering. For example, should listing 2 be realised by integral term recomputation or actuation error feedback, both legitimate and widely used antiwindup techniques in the discrete time domain. Its behaviour would not replicate correctly that of listing 1. As a rule of thumb,

- when starting from a continuous-time block the best policy is to start from its block diagram, including possible nonlinear parts, and to discretise each (linear) component – as $1/(1+sT_i)$ in the example – individually,
- while if one starts from a discrete-time block – e.g., to include a specific product also as continuous-time model – it is advisable to express its diagrams so as to have only one-step delay terms, and then convert to continuous-time by back-applying the discretisation method of choice.

Note also that functionalities easy to express in the discrete time – e.g., automatic/tracking bumpless switch – actually require the corresponding continuous-time model to be switching *stricto sensu*, which is hard to manage for virtually any object-oriented model translator. Experience has shown that the problem can be practically worked around by introducing some additional "very fast" dynamics, exhausting its effect in far less than a sampling period. The only warning in this respect is to adopt solvers that are implicit or at least account for the possibly introduced spurious stiffness, so as to avoid numerical problems. Such features are offered by all the major tools, however. Finally, observe how initialisation is handled. The example should be self-explanatory also concerning its generalisation, but this is another quite critical issue. The same considerations apply to *re*-initialisation when employing both descriptions in the same simulation, as advised for autotuning.

By applying the simple practices just sketched, one can thus model any controller (PID and not only) both as a continuous-time and a digital one, no matter what type of information – the *extrema* being a formal specification only, or a product operation description – is available for the design.

4. APPLICATION EXAMPLES

4.1 Block consistency

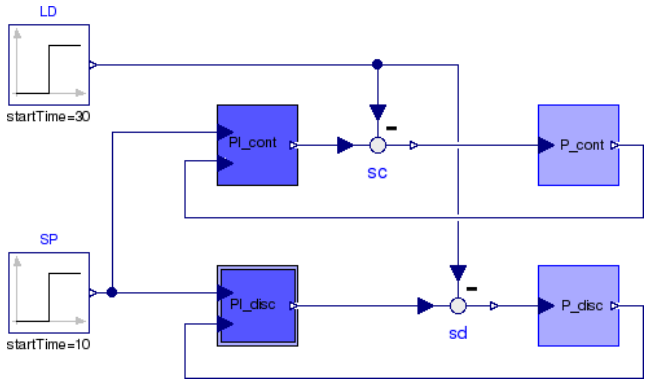

Fig. 4. Modelica diagram for the "block consistency" example.

This example shows whet is meant for "block consistency". A simple loop with the process

$$P(s) = \frac{1}{1+5s} \tag{1}$$

and a PI having $K_p = 6$, $T_i = 3$ is simulated both entirely in the continuous time, and with a digital PI with a sampling time of 0.5 seconds (definitely not quite short). The used Modelica scheme is shown in figure 4.

The outcome of a 60-seconds simulation are shown in figure 5, the applied *stimuli* being a set point and a load disturbance step, the former driving the control signal transiently into saturation.

It can be seen that the correspondence between the two representation is good, which would not be true if the practices sketched above were not adopted. As for efficiency, the entirely continuous-time realisation requires 128 computation steps for the entire run, versus the 1643 of the realisation with the digital PI.

Apparently, having consistent descriptions of controllers in the two domains does allow the analyst to manage the trade-off between simulation speed and detailed representation of the relevant signals and dynamics.

4.2 Autotuning

This example refers to the autotuning PI of Section 3.2; the process under control is described by the transfer function

$$P_1(s) = \frac{1}{(1+s)^3} \tag{2}$$

and the autotuning PI, in both the fully digital (here omitted) and the hybrid versions, is employed with a sampling time T_s of 0.1s, first leading the loop to steady state with a low-performance initial PI, then performing the autotuning operation with a required phase margin of 45°, and finally testing the so obtained PI with a set point and a load disturbance step.

Figure 6 shows the used Modelica diagram, while figure 7 reports the results, proving that the two realisations are *de facto* identical as for their outcome (in both cases, for example, the tuned PI has $K = 1.078$ and $T_i = 1.751$).

On the other hand, however, the number of simulation steps required by the system with the hybrid autotuner in the 240-seconds presented run is 3908, versus the 24007 of the system

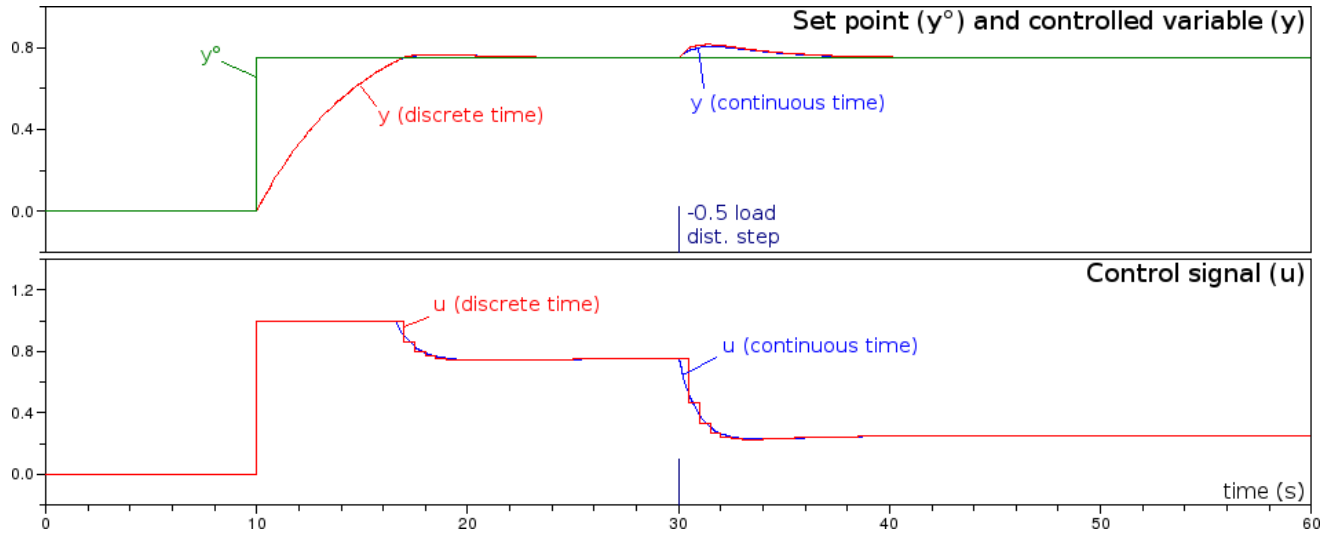

Fig. 5. Simulation results of the "block consistency" example.

with the fully digital one. With so simple a process this does not turn into a significant reduction of the simulation time, but with more realistic (thus computation-intensive) a model of the controlled object, said advantage would – again – be evident.

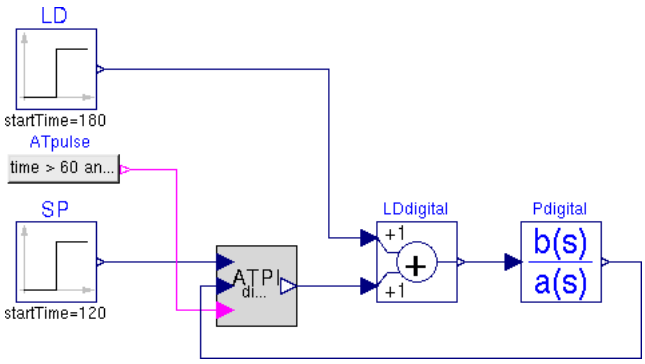

Fig. 6. Modelica diagram for the autotuning example.

5. EXTENSIONS

At present, the library just contains some basic blocks, and its modulating part – the only one shown here – is apparently quite PI(D)-centric.

Of course several extensions are envisaged and some are already underway. Focusing on this point, some specific words on the "extensions" subject are in order, to better clarify how the authors envision this project from an overall standpoint—which of course is just an opinion as the code is being released within the terms of an open license, thus not only derivate works substantiating different points of view are possible, but their creation would be considered a success for the initiative.

Coming back to the main point of this section, apart from the specific subject of this work – that focuses on the consistent co-existence of continuous-time and event-based representation for the same control block – in simulation environments there is undoubtedly a strong need for controller representations that are accurate enough a *replica* of what will then be installed on the real plant. The question, in a view to designing future library extensions, is therefore how far such an idea needs pushing.

In this respect, at least the opinion of the authors is that two main cases can be broadly distinguished.

- The goal is to have an exact representation of what happens with a given control architecture—e.g., a given DCS and/or SCADA by a given manufacturer, or any combination thereof. This requires to develop huge libraries, as rich as the typical menu of industrial control configurations is, replicating exactly the choices of the considered manufacturer(s). No doubt this has a practical relevance, but no doubt either this is outside the scope of the project presented here. Users willing to undertake a work like that just described, however, can find the continuous-time models ready, and get inspiration from their event-based counterparts as for the way to go when replicating the particular algorithms they need to consider.
- The goal is to determine which characteristics a control architecture and/or algorithm should possess in order to properly realise a given control strategy described in the continuous-time domain. This may happen for example when a custom controller has to be designed, which is not infrequent e.g. in the embedded systems' domain, and one has to choose the processor, size the converters' resolution, write and test the digital algorithm, and so forth. Here too, interested users can take profit of the presented work in more or less the same way as in the previous case, with the only difference that a small number of specific blocks need developing instead of a vast library of general-purpose ones.

Quite intuitively other situations can be envisaged, but the two just mentioned appear to be the most interesting ones. In a view to facilitate both library uses without requiring too much time to sufficiently master the library itself, the decision was thus taken to expand the existing *nucleus* including other basic control blocks such as the lead-lag, the fixed time delay or similar ones, but not to include a large set of pre-assembled control structures like e.g. the cascade one. Such structures can be in fact realised with the set of basic blocks that will be present in the library after the present developments are complete.

On the other hand, plans are to expand the set of *autotuning* methods and procedure. According to the authors' experience, in fact, in the majority of cases where the introduction and

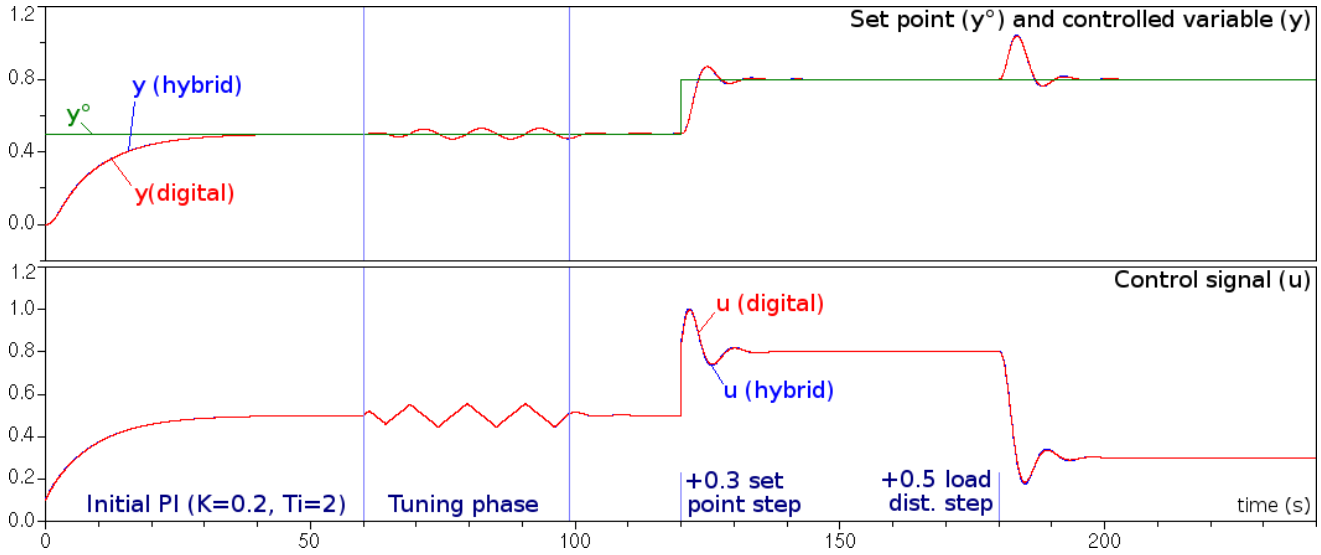

Fig. 7. Simulation results of the autotuning example with both the hybrid version presented and the fully digital one.

use of autotuning were not successful, a significant part of that undesired outcome is to be blamed on an incorrect choice of the tuning procedure. Such a choice is however sometimes tricky, as it consists not only of choosing a "good" tuning *method* and an "informative" way of providing control specifications, but also of accounting for the process stimulations that will then be used to gather the required data, considering issues such as actuator limits, noise, possible outliers and so forth.

Having the possibility of experimenting not only with autotuning *methods* in control analysis environments, but also with autotuning *procedures* coupled to accurate process simulators, is in the authors' opinion an effective way to foster a correct use of the autotuning technology, let alone a deeper comprehension of it and its application for users who may also need to set up, and not just to select and use, autotuning controllers.

Finally, in the light of the development line just sketched, effort will be spent to expand the library section devoted to logic control – not treated here given the scope of this particular work – and its integration with the modulating one.

6. CONCLUSIONS

In this paper a library of controller models was presented. The main peculiarity of the library is that controllers are represented both as continuous-time models and as digital ones. The two representations are kept consistent and the user can simulate the behaviour of the controllers with both of them.

This allows, for example, to design a controller based on its continuous time representation and to assess its event based implementation behaviour, using in each case (or any combination thereof if the system comprises more than one controller) an overall model the complexity of which is tailored to the study at hand.

A key role in the library is to be played by autotuning, which was here initially considered, and for which the co-existence of the mentioned two representation is particularly beneficial from the standpoint of simulation efficiency. Some tests were reported, both to show that the two representations are kept consistent, and to present an autotuning PI exploiting the proposed modelling approach.

The presented library will be in the future extended to form a larger one, since at present it is mainly focused – as far as modulating control is considered – on standard PI and PID regulators.

Guidelines for such extensions were provided, but the authors also hope that this project may encourage and stimulate cooperation, so that different points of view can be explored.

REFERENCES

Åström, K. and Hägglund, T. (1984). Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, 20(5), 645–651.

Åström, K. and Hägglund, T. (1991). Industrial adaptive controllers based on frequency response techniques. *Automatica*, 27(4), 599–609.

Åström, K. and Hägglund, T. (2006). *Advanced PID control*. Instrument Society of America, Research Triangle Park, NY.

Besançon-Voda, A. and Roux-Buisson, H. (1997). Another version of the relay feedback experiment. *Journal of Process Control*, 7(4), 303–308.

Casella, F. and Leva, A. (2006). Modelling of thermo-hydraulic power generation processes using Modelica. *Mathematical and Computer Modelling of Dynamical Systems*, 12, 19–37.

Dorf, R. and Bishop, H. (1995). *Modern control systems*. Addison-Wesley, Reading, UK.

Leva, A. (1993). PID autotuning algorithm based on relay feedback. *IEE Proceedings-D*, 140(5), 328–338.

Leva, A. (2005). Model-based proportional-integral-derivative autotuning improved with relay feedback identification. *IEE Proceedings - Control Theory and Applications*, 152(2), 247–256.

Luyben, W. (2001). Getting more information from relay feedback tests. *Ind. Eng. Chem. Res.*, 40(20), 4391–4402.

Mattsson, S., Elmqvist, H., and Otter, M. (1998). Physical system modeling with Modelica. *Control Engineering Practice*, 6, 501–510.

Yu, C. (1999). *Autotuning of PID controllers: relay feedback approach*. Springer-Verlag, London.