A Modelica Library for Industrial Control Systems

Marco Bonvini Alberto Leva
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133, Milano, Italy

Abstract

Many studies for which simulation is necessary include the presence of control systems. While plenty of Modelica libraries are nowadays available to accurately represent the plant, the same is not so true as for the control elements, since industrial ones are endowed with a number of functionalities – and often present system- or even vendor-specific peculiarities – that are not represented by the typical blocks (e.g., based on transfer functions) offered by the existing libraries. This paper is an attempt to start filling the gap and provide an efficient solution, structured and organised in such a way to be easily understood by control specialists, and to ease information transfer between simulation studies and implementation.

Keywords: industrial controllers, simulation

1 Introduction

In many simulation studies, control plays a relevant role. Sometimes this is because the study is precisely aimed at setting up the control system for the plant at hand, but in many other situations, even if control synthesis is not the main goal of the study, the behaviour itself of the modelled object depends significantly on the operation of some controls. As such, quite often the representation of the control system deserves substantially the same accuracy as the representation of the physical plant (in the broad sense of the term).

At present, numerous Modelica libraries are available to represent plants with a virtually arbitrarily accuracy, but the same is not true – at least, to the best of the authors' knowledge – for controllers. To appreciate that, the interested reader could for example throw a glance at the PID block as provided by any control environment, be it targeted to a PLC, a DCS, or whatever. Most likely, he/she will see something similar to the two examples shown in figure 1.

Apparently, such blocks are more articulated than for example the PID of the Modelica Standard Library

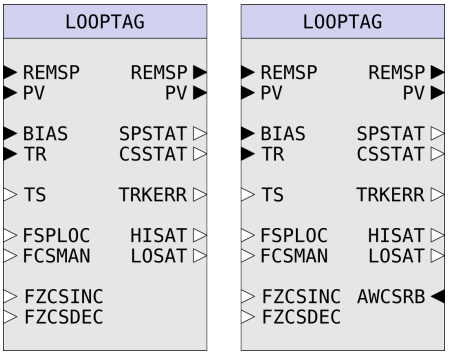

Figure 1: Two examples of PID blocks as sen in typical industrial control tools.

(MSL)—as by the way real-life control systems do exhibit a number of peculiarities that are not accounted for in "textbook" representation, see e.g. [9]. The remarks just made are in no sense meant to be a criticism, it is worth stating; nonetheless they evidence that for the simple controller representations of the MSL (or analogous ones) to be adequate, some conditions are necessary. Summarising, and sticking to the PID example,

- the specific form of the controller (let alone the detailed operation of the control algorithm) must not be relevant for the problem,
- and the operation of typical elements of industrial controllers, such as tracking and locks, must not be of concern either.

If this is the case, MSL-like representations are perfectly adequate. If on the contrary either this is not the case in the simulation *scenarii* to be considered, or one wants to describe the control system so as to be capable of simulating the controlled plant in its entire set of operating modes, the same representations cannot serve the desired purpose.

For the reasons above, and after several years during which the authors and their group have been developing *ad hoc* solutions for individual cases, the decision

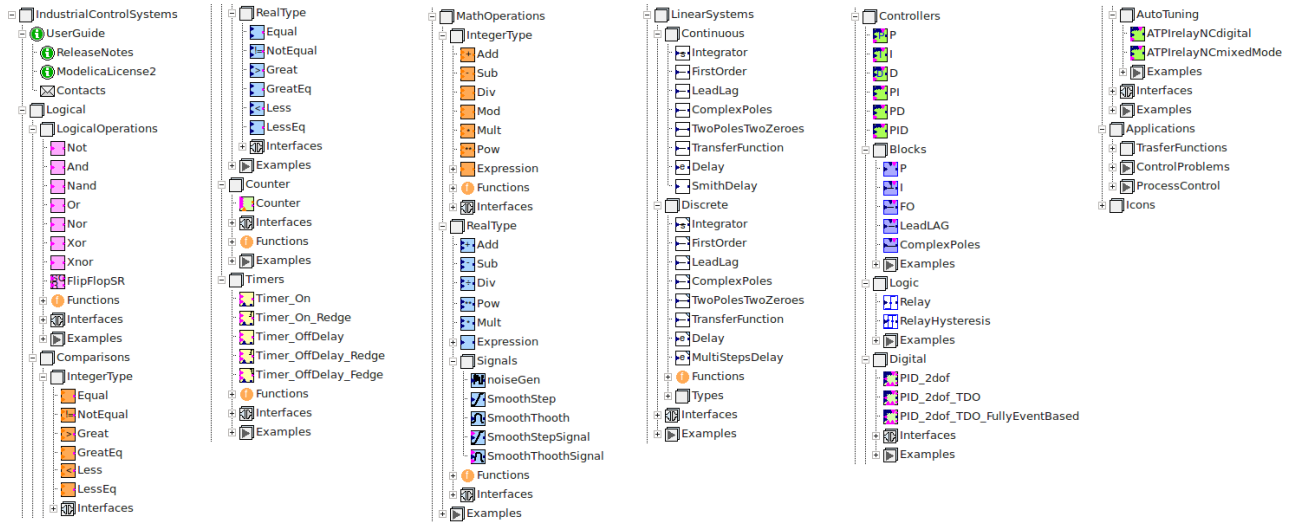

Figure 2: An overview of the library structure.

was recently taken to put all of that knowledge and Modelica code together in a structured manner.

The result is the library described in this paper, which is organised as follows. Section 2 presents and motivates the most qualifying characteristics of the library. Section 3 presents the library structure giving just a quick overview, as the library documentation provides full detail on the matter. Section 4 reports some simulation examples, these too available in the library, to evidence and further motivate its distinctive characters previously discussed. Finally, in section 6 some conclusions are drawn, and future work is sketched out.

2 Main characteristics of the library

To fulfil the requirements envisaged in section 1, it is first necessary to include both modulating and logic control elements.

For modulating elements, it is required to account for the typical representations of the major control blocks – see e.g. [8, 3] for how many forms a PID can take – and the typical realisations of the main nonlinear functionalities: for example, taking again the PID as example, antiwindup can be realised internally or by reading back the applied control from the actuator. Also, logic functionalities need incorporating, such as tracking and the possibility of preventing the control signal from increasing or decreasing, which is of great usefulness in cascade controls. Finally, different algorithmic realisations (e.g., positional or incremental) need considering, since in some cases they can affect the behaviour of the element, especially if controller parameters can be modified online as is the case for gain-scheduling blocks.

For logic elements, the typical set available in SCADA-like products needs representing, including timers, counters, sequencers, and so forth.

Then, it would be advisable that the modelled control elements allow for variable-step simulation, to avoid obliging the analyst to use the library only with fixed-step solution, which could be unacceptably inefficient in more than one case. As such, the choice was made to provide both a time-driven and an event-driven version of the same element wherever this is possible, and research is underway to extend this coverage to the whole library.

Moreover, in a view to good acceptance and wide utilisation, care was taken to give the library elements a look and feel as similar as possible to what a user of SCADA (or analogous) tools expects to see. This was not pursued up to its extreme consequences, but is definitely a peculiarity.

Finally, an initial set of autotuning controllers is included, building on previous research see e.g. [1, 5, 7, 6]; this is meant both to ease control setup in simulation, and to help the user familiarise with that technology, and the underlying theory.

3 The library structure

The library is organised into subpackages; a list of the major ones is given below.

DOI 10.3384/ecp12076477

- **Logical**, that contains all logical elements, timers, counters, and so forth.
- **MathOperations**, including the necessary operators for real and integer numbers (which is sometimes very useful to correctly represent the operation of some industrial blocks).
- **LinearSystems**, where some blocks are contained that can be used to easily close loops to test controllers. Part of those blocks are also related to well known controller benchmarks, see e.g. [2]; of course this subpackage is provided basically for convenience and to obtain a self-contained library, but many alternatives can be used.
- **Controllers**, where both modulating and logic control blocks are represented, in three basic (and interchangeable) manners: (a) as continuous-time equations, (b) as equations but evolving by events, and (c) – when multiple assignments could not be avoided, although research to solve this is underway – as algorithms.
- **Applications**, that contains a quite large set of examples to better understand and use the library.

Figure 2 shows an overview of the library structure. Readers that are familiar with control systems and control theory will easily get familiar with the library and its structure (just by observing the library components); non experienced user will find further details into the included documentation.

3.1 Interfaces

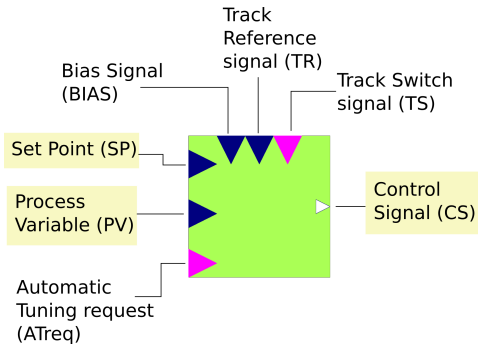

Figure 3: Interface for a generic controller. The input/output connector evidenced in yellow are always present, the other ones can be conditionally selected.

Each model/block/controller contained into the Industrial Control Library can be connected together with other models ones through its standard connectors, defined in the Modelica Standard Library. In each subpackage, an ad-hoc partial interface model has been defined in order to improve the readability of the code, and reduce as much as possible the number of code lines spent for non specific purposes. Figure 3 shows the interface of a generic controller. The input/output connectors of such a block can be conditionally selected through various boolean flags as shown in table 1. With these conditional connectors a controller can be used even if it does not use all its features, without connecting dummy inputs to it and thus increasing the clarity of the control scheme. The interfaces and the variables of the models have been named according to the standard terminology in the field of control systems. The interested reader that is not familiar with this topic can find more information in [4].

Table 1: This table contains the definition of the interface of a generic controller with its conditional input/output connectors.

Name	Description	Conditional?
SP	Set Point	NO
PV	Process Variable	NO
CS	Control Signal	NO
TR	Track Reference	YES
TS	Track Signal	YES
Bias	Bias signal	YES
ATreq	Automatic Tuning request	YES

4 Simulation examples

This section contains a small sample of the examples contained in the library, to show the possible usage of some models, and also evidence the usefulness of adopting the proposed representation.

4.1 Zero crossing count

This examples uses some blocks of the Logical subpackage, as shown in figure 5. More in detail, the signal represented by $y(t) = sin(t)$ is compared with to $z(t) = 0$. Each time the signal crosses the reference, the boolean output of the comparison block rises. The rising edges are counted by the digital counter, in the period comprised between $t = 2.2$ and $t = 10.2$. Figure 5 reports the Set and Reset Count signals, while figure 6 shows the behaviour of the counter value.

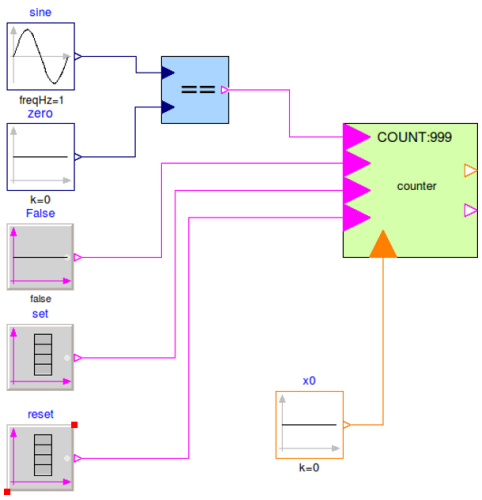

Figure 4: Scheme of the zero crossing count model.

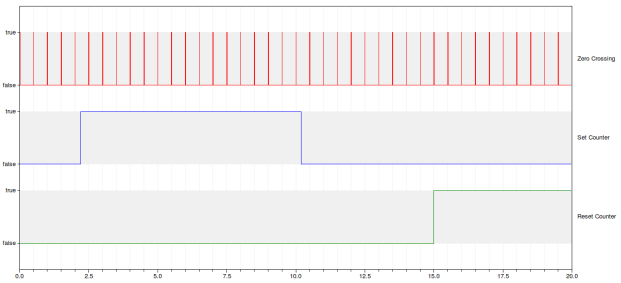

Figure 5: Zero crossing signal, Set count signal and Reset count signal.

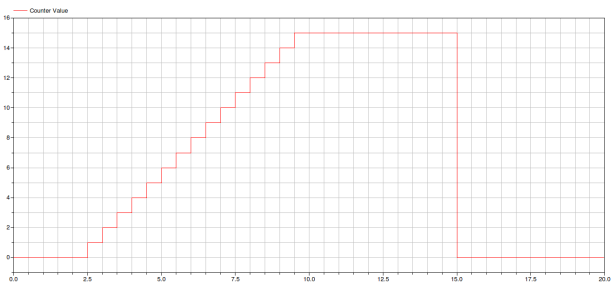

Figure 6: Counter value.

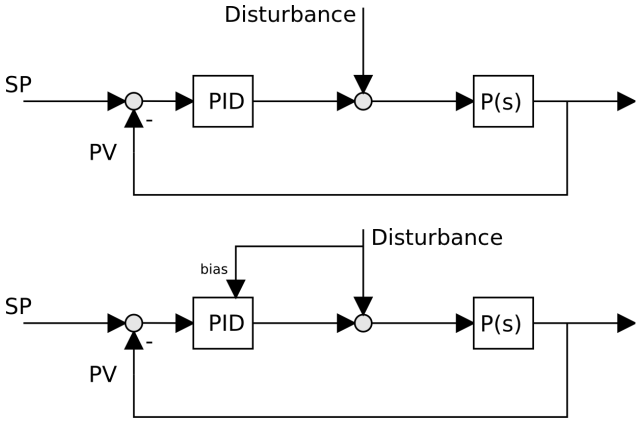

Figure 7: Classic PID controller: without bias signal (top) and with bias signal (bottom).

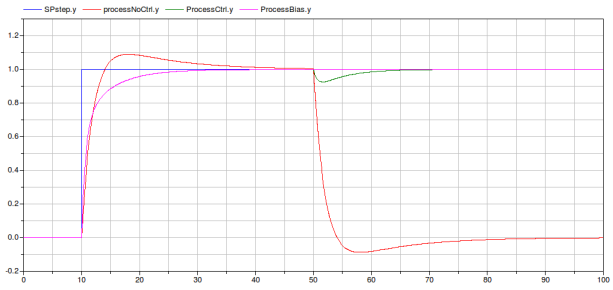

Figure 8: Set Point (blue), Process Variable without control (red), with a PID controller (green) and with a biased PID controller (magenta).

4.2 PID with bias and tracking mode

In this example the second order process defined as

$$P(s) = \frac{(1+15s)}{(1+2s)(1+10s)} \qquad (1)$$

is controlled by a PID regulator, to track given step Set Point signal, and reject a load disturbance acting on the process input (as shown in figure 7). Two controllers are compared, namely a PID and a PID with bias input. Figure 8 reports the Set Point (blue line), the Process Variable of the process without control (red), controlled with the PID (green) and the PID with bias (magenta). The PID rejects the disturbance just via the feedback path, that makes its action slower. On the contrary, the PID with bias acts immediately, thanks to its feed forward character.

Carrying on to representing the tracking mode operation (see figure 9), an example is shown with the process defined in (1), still controlled by a PID. Figure 10 shows the Set Point, the process variable and the tracking switch signal, while figure 11 shows the control signal, the track reference and the integral action of the controller. The Tracking mode starts at $t = 40s$, before the controller has led the process variable to the Set Point reference. When the Tracking mode starts, the control signal becomes equal to the track reference (as shown in figure 11). In this case the track signal decreases and then increases, moving the process variable in a neighbourhood of the set point. When the tracking mode is enabled, the integrator does not integrate the error signal, rather is managed in such a way to be consistent with the track reference. Thus, the transition from the tracking mode to the automatic one is bumpless.

DOI 10.3384/ecp12076477

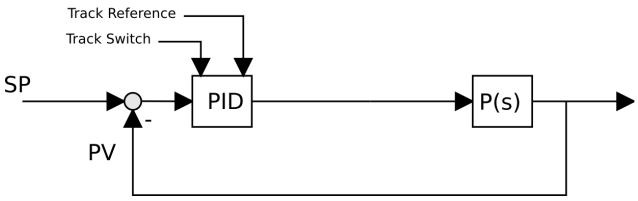

Figure 9: PID with Track Switch ans Track Reference signals

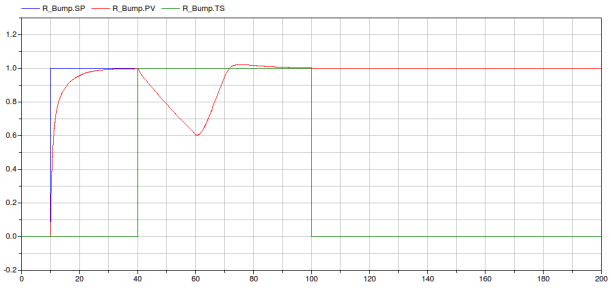

Figure 10: Set Point (blue), Process Variable (red), and Track Switch signal (green).

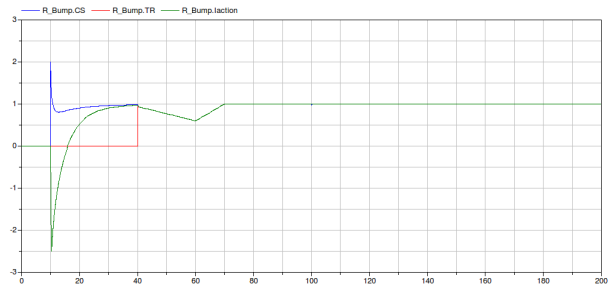

Figure 11: Control Signal (blue), Track Reference Signal (red) and Integral action (green).

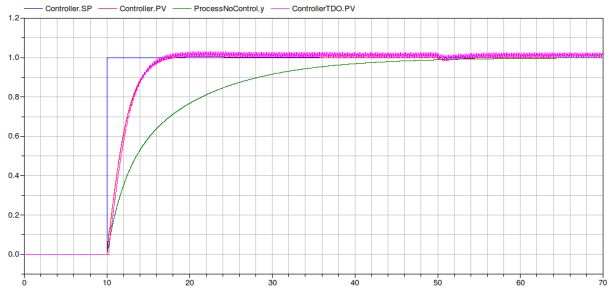

Figure 12: Set Point (blue), Process Variable without control (green), process Variable with PID (red) and PV with TDO PID (magenta)

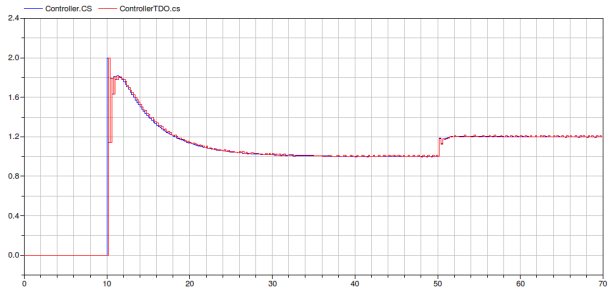

Figure 13: Control Signal (blue) and CS of TDO PID (red)

4.3 Time Division Output controller

The process (1) is here controlled with a Time Division Output PID. Such an actuation scheme is used to have an on/off actuator behave like a modulating one, and is quite typical when either the actuator cannot be partialised, or doing so would unacceptably reduce its efficiency. The controller, implemented in its digital algorithmic form, first computes the control signal, and then converts it into the duty cycle of a rectangular wave of assigned period. Figure 12 contains the Set Point reference (blue), the Process Variable of a process without control (green), the Process Variable of a digital PID (red) and the Process Variable of a TDO PID (magenta). Since the TDO control signal changes continuously, the relative process variable has a sort of ripple, however the overall behaviour is essentially the same as the digital PID without TDO. The control signals computed by the two controllers are shown in figure 15.

4.4 Cascade control with increment and decrement locks

This examples compares two cascade control schemes, one with and one without increment/decrement locks. When two controllers are connected together in a cas-

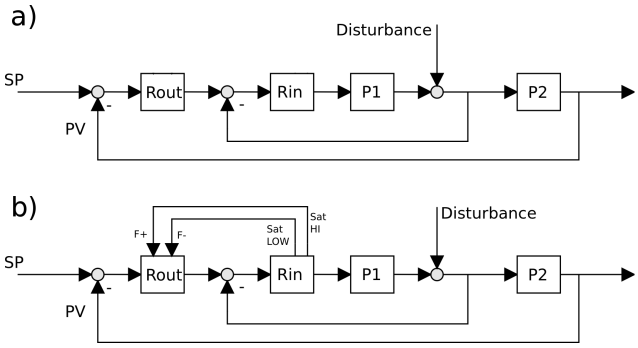

Figure 14: Cascade control schemes: a) without increment/decrement locks – b) with increment/decrement locks.

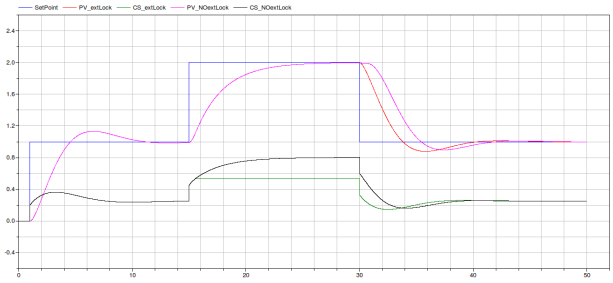

Figure 15: Cascade control with and without increment/decrement locks – outer set point and process variable, inner set point.

cade control scheme, the inner controller typically regulates the actuator, while the outer one provides the Set Point reference for the inner one. Since the inner controller acts on the plant, its Control Signal has to be limited, and AntiWindup is in order, but in general it is not possible for the *outer* controller, to know the values for which the *inner* regulator saturates.

Such a problem can be avoided by using the PID in its incremental form, using the Increment/Decrement lock feature, and creating an external (logical) loop between the controllers, as shown in figure 14.

If the inner regulator saturates, its satHi signal becomes true. Connecting this signal to the forbidIncrement input of the outer controller, avoiding a useless and potentially dangerous increase of its Control Signal (that is the Set point of the inner controller that saturated). With such a scheme, the mentioned inter-loop windup-like effect can be avoided.

In figures 15 and 16, that show the results, the green line is the CS of the outer controller with Increment/Decrement lock, while the black one is the output of the outer controller without Increment/Decrement lock. The black line shows a windup like effect that turns in a slower reaction when the Set

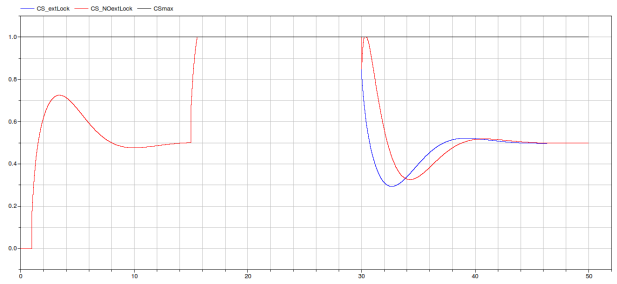

Figure 16: Cascade control with and without increment/decrement locks – inner control.

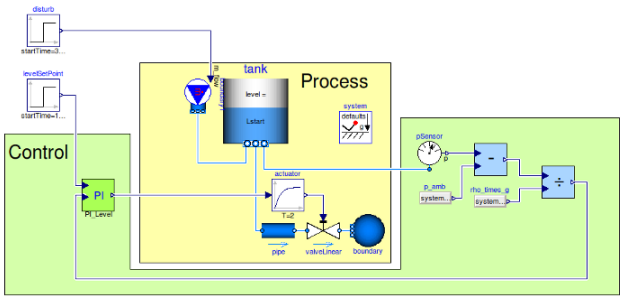

Figure 17: Modelica diagram of a level control scheme. The two subsystems (the control system and the process to be controlled) are evidenced with different background colors.

Point changes at time $t = 30$.

4.5 A level control case

In this example, models from the presented library are used together with models from the MSL. The aim of this example is to show the usefulness of the presented models, and how they can be easily integrated and connected with others. For this purpose, the chosen example refers to the problem of controlling the water level in a tank. The water level is the process variable, and the system (see figure 17) is composed of a tank and a pipe connected to a valve, that discharges water to the atmospheric pressure. The valve actuator is simply represented by a first order system with unity gain.

The control system is composed of a measurement part and a control (*stricto sensu*) one. Concerning the measurement part, the pressure sensor measures the absolute pressure at the bottom of the tank. The measured pressure p_m is subtracted from the atmospheric pressure p_0, and then divided by the gravity acceleration g and the water density ρ, in order to obtain the water level

$$l = \frac{p_m - p_0}{\rho g} \tag{2}$$

The PI controller, given the level measurement and

DOI 10.3384/ecp12076477

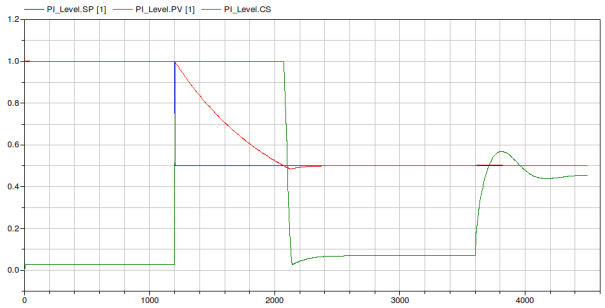

Figure 18: Set Point water level reference, Process Variable and valve position command

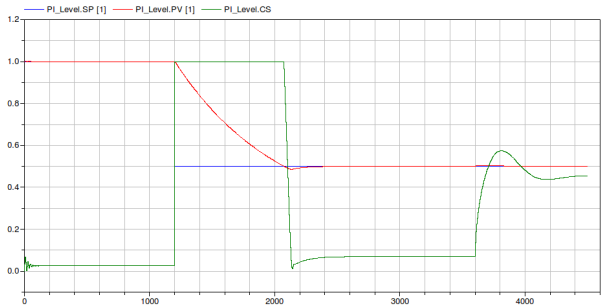

Figure 19: Set Point water level reference, Process Variable and valve position command (discrete time controller, $Ts = 5$ s)

the set point, computes its control action, i.e., the prescribed valve position, limited between $CS \in [0,1]$ in order to avoid windup effects (thus $CSmin = 0$ and $CSmax = 1$). The tank is $2m$ height, and the water level at time $t = 0$ is $L = 1$ m. In the first phase the controller is required to maintain the level at the initial value ($SP = 1$ m), while at $t = 1200$ s the level set point has a steep variation ($SP = 0.5$ m). The controller has to act on the valve in order to decrease the water level to the desired value. A disturbance, represented by a water mass flow rate entering the tank, becomes different from zero at time $t = 3600$ s. Figure 18 shows reference, water level and valve position command.

The simulation can be performed at an initial stage assuming that the controller is a continuous time one ($Ts = 0$), and the math operations are in double precision (FixedPoint = false). In such a phase, it is thus possible to concentrate on the controller design (not on implementation-related facts).

As a further stage, one could introduce more details in order to simulate a more realistic system. At first it is possible to introduce the time discretisation, and investigate the effects of the sampling time. Figure 19 shows the simulation results with a sampling time $Ts = 5$ s.

Figure 20: Set Point water level reference, Process Variable and valve position command (discrete time controller, $Ts = 5$ s and Fixed Point math operations)

An additional level of detail can be the introduction of fixed point math operations. In this example, a number of bit $Nbit = 24$ was chosen, which means that the integer number that can be represented are comprises between $MIN = -8388609$ and $MAX = 8388608$. At a first stage, the measured pressure have to be subtracted of the ambient one. In the worst case, the higher pressure value that can be read as input from the math operation block is $101325 + 1000 \cdot 9.81 \cdot 2 = 120945$, that is more or less two orders of magnitude less than the higher integer number MAX. This means that input numbers can be multiplied by a scale factor comprised between 10 and 50. In this case the scale factor has been chosen as $sFactor = 20$. In a similar way, the scale factor for the division can be chosen (In this case, $sFactor = 500$). Note that a large number of bits is required because the pressure variation is small with respect to its absolute value. Using such a modelling approach, it is possible to estimate the amount of bits required, and to directly test the correctness of the design strategy. Figure 20 shows that the numerical errors due to a wrong design are visible on the Control Signal.

5 Towards Modelica 3.3

The recent definition of the version 3.3 of the Modelica language introduces new elements for describing synchronous behaviours, and also new elements suited to define synchronous state machines. This evolution is primarily made to ease the activity of modelling realistic control algorithms.

These evolutions will introduce some advantages in the development of models that are pure discrete or logical, since a standardised framework for developing such models will help in the design, creation and maintenance of models in which many of these com-

ponents are connected together. Considering elements that can be either continuous time or discrete time, and which events are not regular but can be dynamically driven; however, it is not yet clear if this language evolutions will fit also such model characteristics, that (as shown) are of great importance for tailoring the simulation burden to the needs of the addressed study.

The last interesting point that has not yet been considered, but in the authors' opinion should be, is the introduction of the Fixed Point arithmetic. The presented library takes into account this problem and it is managed in a preliminary and simplified way, providing a solution just for simple cases. The introduction of a new type of variable with its specific operations will be an important step in the direction of a really control (and control synthesis) oriented simulation tool.

6 Conclusions and future work

A Modelica library for industrial controllers was presented, with several peculiar features, and some examples were shown to illustrate its potentialities.

In the authors' opinion the library can significantly help the analyst who has to address studies where a precise control representation plays a relevant role—more frequent a case than one may expect at a first glance, by the way. The presented library in the first place responds to such a demand, and in addition tries to preserve the advantages of variable-step simulation when possible—a matter on which further research is however underway. The library is by definition extensible, so that one may even want to include the exact (i.e., code *replica*) representation of some block of interest, employing those already realised as a starting point. Implicitly, then, the library has also a didactic value, since the user can see how several concepts are actually put to work. Some examples were reported to show the library operation. All of these – plus others omitted here for space reasons – are available in the library itself (available at http://home.dei.polimi.it/leva/download.html), for the convenience of the interested reader.

Future activity (apart from the already mentioned one related to simulation efficiency) will be directed at expanding the library in all its sections, including the autotuning one, and to extensively use it in simulation studies. The community is encouraged to use, improve – and correct if necessary – the library, and feedback would be highly appreciated by the authors in order to continuously improve the results.

References

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

[2] K.J. Åström and T. Hägglund. Benchmark systems for PID control. In *IFAC Workshop on Digital Control – Past, present, and future of PID Control*, Terrassa, Spain, 2000.

[3] K.J. Åström and T. Hägglund. *Advanced PID control*. Instrument Society of America, Research Triangle Park, NY, 2006.

[4] W. Dunn and W.C. Dunn. *Fundamentals of industrial instrumentation and process control*. McGraw-Hill Professional, 2005.

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

[6] A. Leva and M. Bonvini. Efficient hybrid simulation of autotuning PI controllers. In *Proc. 8th International Modelica Conference*, Dresden, Germany, 2011.

[7] A. Leva, S. Negro, and A.V. Papadopoulos. PI/PID autotuning with contextual model parametrisation. *Journal of Process Control*, 20(4):452–463, 2010.

[8] A. O'Dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific Publishing, Singapore, 2003.

[9] F.G. Shinskey. Process control: as tought versus as practiced. *Industrial & Engineering Chemistry Research*, 41(16):3745–3750, 2002.

DOI 10.3384/ecp12076477