Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

A control education software suite to bridge methodological and engineering aspects

Alberto Leva * **Chiara Cimino** ** **Silvano Seva** **

* *DEIB, Politecnico di Milano, Italy*
(e-mail: alberto.leva@polimi.it)
** *PhD student at the DEIB*
(e-mail: {chiara.cimino,silvano.seva}@polimi.it)

Abstract Software obviously plays a relevant role in both control education and engineering. Methodology-centred tools help learning in the class and are also useful for high-level tasks in the profession. Engineering-centred ones are necessary for any realistically sized problem, and their comprehension helps designing efficient and well maintainable control applications. However, brutalising for brevity, the two sets of tools hardly intersect one another, and are most often addressed with very different educational viewpoints—and frequently with some under-emphasis on the engineering side if not for laboratory practice. This paper provides a reasoned overview of the scenario just sketched, and based on a long experience, proposes a coordinated set of tools, selected among well assessed and solidly maintained ones, to help the students bridge methodological and engineering aspects into a unitary *forma mentis*.

Keywords: Control education; Engineering education; Control software.

1. INTRODUCTION

Suppose to ask a control faculty and an industrial control professional to list what he/she considers to be "software for control". The answers will most likely be quite different—and both correct in their context, to be clear about this right from the beginning.

As a reasonable hypothesis, the academic would almost certainly include MATLAB and Simulink – or free software alternatives like GNU Octave or Scilab – as the central item. If the taught course involves advanced mathematics, he/she could also mention some symbolic computation tool like Mathematica (or free equivalents like wxMaxima). Interactive tools and learning modules, based e.g. on SysQyuake or Easy Java Simulations, might be included as well. If the course is connected to some specific domain (e.g., chemical) some modelling and simulation software specific to that domain could appear. Programming tools for controllers like PLCs, or more hardware-agnostic such as LabVIEW, would be normally considered only if there is a laboratory activity, and be seen as substantially confined to that activity. If there is a focus on microcontrollers, he/she could also mention the Arduino and its IDE, or analogous products, and possibly envisage some use of programming languages like C—here too, if there is some laboratory. General-purpose modelling and simulation environments like Modelica would be seldom considered, while tools to manage an articulated project – for example, development environments for distributed industrial control systems and/or SCADA – would in general be practically absent.

Imagining the response of the industry professional is more difficult, but we nonetheless try an educated guess to exemplify the viewpoint differences. The focus would be most likely on SCADA and control development environments including HMI building, IEC 61131-3 languages, distributed systems and the IEC 61499 standard [1], field busses and communication including Modbus, CAN, EtherCAT, Profibus, Profinet and the like, possibly C programming if dealing with embedded systems, HIL/SIL platforms and assessment tools. MATLAB would probably be mentioned, but when modelling, analysis and simulation is involved, quite frequently the focus would be set on some domain-specific tool to manage as much as possible of the entire project, possibly connected – especially for large businesses – to some PLM and/or asset management environment.

We have to admit that in the above sets of software we have made the discrepancy probably a bit sharper than it can be in many cases, but no doubt it exists. And to evidence that the "theory/practice" equilibrium – whatever is exactly meant for that– can easily approach marginal stability, we could observe (based also on experience) that should the question be just slightly nuanced so as to read "what software should we use in control teaching", the industry viewpoint could not so unlikely move toward "what control software we should teach".

Indeed, the "software for control teaching" matter is delicate in more than one sense. On one hand, since education is education and training is training, we believe that the academia should not teach the details of any software—on the contrary, graduates should be able to exploit their conceptual knowledge and withstand any change in the particular tools to apply. On the other hand, however, we also believe that the students should exit their classes with clear enough ideas about how a realistically sized control

[1] Information about IEC standards can be found on the International Electrotechnical Commission web site `https://www.iec.ch/`.

project is managed, to avoid being overwhelmed by the huge amount of technological training they will need (and receive) when entering the profession, and to preserve the methodological capabilities that they will soon discover to be extremely precious.

In this paper – continuing the research presented in Maggio and Leva (2011) and in Leva and Cimino (2019), that focused respectively on education about control code creation and about control applications organisation – we attempt to join the two needs just mentioned with one another, up to a reasonably feasible extent, by proposing and motivating a set of tools and their coordinate use as designed through about 20 years of didactic activities. We believe (and hope) this set and the consequent suggestions to be a useful complement for the wealth of software centred on control education that our community has been developing along the years.

2. LITERATURE REVIEW AND MOTIVATION

In the literature, the industrial viewpoint on control education – as well as the outcome of that education in the professional world – have been receiving a long-lasting attention (Bristol, 1986; Vyatkin, 2013) from the control but also from the computer community.

Restricting the focus to the scope of this paper, besides just employing control-oriented mathematical software within automation courses (Wenjiang et al., 2009), in the literature about software for control education three topics emerge with particular strength. The first one may be called "interactive learning tools" and offers valid contributions like e.g. Guzmán et al. (2006) for PID control, with network-based technologies also enabling the creation of complete applications (Gonzalez et al., 2012; Dekemele et al., 2018); an overview on the matter can be found in Guzmán et al. (2016).

Here we shall not be talking about interactive modules and subject-specific (e.g., on PID) tools. This must in no sense be interpreted as diminishing their importance. We love them, show some in the class, and encourage the students to use them for autonomous learning. We simply view these tools as part of the methodological rather than technological side of control education, hence already well covered by many works other than of this paper.

The second major topic we mention is laboratory equipment and the relative software. As witnessed e.g. by Coelho et al. (1997); Leva (2003); Montesinos Miracle et al. (2007); Tekes et al. (2018) this is a long-standing research, receiving in recent years impulse by the vast availability of microcontrollers or embedded computers at large (Candelas et al., 2015; Hoyo et al., 2015; Omar, 2018).

Here we shall not be talking about didactic laboratory equipment/software either (if not in Section 5 to relate them to the presented research). Again, no intention to diminish their importance. We use small experiments to get students in touch with their first problems and see that models do represent the physical world. But from the technological side the environment is highly idealised with respect to the real plant floor, and most relevant for us, the process of taking controller models and getting to operational software is either totally hidden, or in general so "in the small" to not require – hence not use – any structured development methodology or standard. Simplifying for brevity, we view these tools as useful for a physical introduction to feedback, but as they are most typically configured, not attacking the core of (software) control technology education needs.

The third topic is virtual/remote laboratories. on which a general viewpoint – including bibliometric impacts – can be found in the survey by Heradio et al. (2016). Within this topic there are experiences involving PLCs (Bellmunt et al., 2006) and also communication standards (González et al., 2016), but in general the student is involved into the software generation process – in the remote case for quite apparent reasons – up to a very limited extent.

Here we shall not address virtual/remote laboratory, for reasons that should at this point be evident. We end this brief review by signalling for completeness an interestingly peculiar approach – somehow an exception to the above but quite isolated – where visual realism in the representation of the plant floor is made central (De Magalhães et al., 2011) and that turned into commercial PLC-centred didactic products (Riera et al., 2009).

Summing up, our point is that the literature contains a lot about software (and laboratories) to teach control, but much less on teaching how software is used to make controls—meaning complete systems with communication, timing, HMI and so forth. This is not a criticism, just noticing a fact. The available tools serve their job well, but consistently, do not address control software technology if not marginally as for the underlying engineering culture. In a nutshell, we can thus evidence three major resulting cultural problems for the students.

- Scarce ability of coordinating various software tools toward a common objective, using the right one for the right operation. This can later on result, in the professional environment, in the quest for "the one tool making all and only what I need". We would like to counteract such a mental attitude, making the students – and prospective engineers – not look for a toolbox to fit their needs, but also learn to compose and use – and above all improve as technology progresses – their own box of tools.
- Tendency to use standard development tools like e.g. the IEC languages just as programming ones, often showing poor capabilities of correlating them to the way the addressed control system was designed and analysed, so that the correlation between the final programs and the models that originated them is not easy to see—if not just destroyed by choices that one *thinks* to be just programming ones, while in fact they modify the behaviour of the controller with respect to what its model said. In the long run, this could generate the idea that models in the end are not useful to create a control application, and what really counts is programming ability, because "the model is the code". We would like to contrast this problem as soon as possible.
- Poor ability to work with large projects, in general and in the particular cases in which feedback needs placing into context. Since not organising and maintaining a large code base properly is a major source

of inefficiency in the development of software, and in the case of control also a source of many undesired behaviours, not curing this issue could in the long run further strengthen the idea that in control classes "they would better have taught me how to manage a project in the [put any product name here] development environment". The need for preventing such a mentality is apparent.

Our intention with the proposal shown herein is not to attempt providing in control courses a culture that requires years of experience; this would be nonsense. We however deem it feasible to at least complement courses with a non episodic or just laboratory-confined presence of industry-oriented control (software) technology, to help the students acquire a solid grasp on that technology in a strong relationship with their methodological competences.

3. THE NEEDS

We assume that the students we are considering receive adequate education on the systems and control theory. This is a prerequisite for any successful application, is not the subject of this paper, and cannot be replaced by any training "by doing", i.e., based on acquiring skills to carry out specific operations. We also assume that they know about writing physically grounded models—a skill apparently on the "methodological" side of the matter in the sense of Section 2.

This firmly said, however, verifying that the student can perform a certain set of operations *after being taught more the underlying principles than the operations themselves*, is certainly a good means to assess that education about those principles was successful. The role of software in this context is to provide the tools for carrying out the said operations—and since these shall inevitably be tied to the activities of a control engineer, also to put the student in contact with how software is used in the profession.

We start defining our needs by devising a (non exhaustive) list of operations in the sense above. As the set of tools we are then proposing is meant to be used through all the education of a student and beyond, the list is not ordered by level (undergraduate, graduate and so on).

(1) Starting from a model of the plant to control, possibly nonlinear and hybrid (e.g., differential equation plus automata)
   (a) linearise and express the transfer function(s) of interest symbolically,
   (b) set up and assess the required modulating controls specifying them – no matter how obtained – as continuous-time block diagrams,
   (c) set up and assess logic control models, for example as Petri nets,
   (d) turn these models into logic controller specifications in standard IEC languages;
(2) convert an IEC (e.g., SFC) control specification into procedural code (C, C++, ST, and so on);
(3) convert a block diagram into procedural code;
(4) write simulation codes for (simple enough) plant models in procedural code;
(5) realise modulating and logic controls with professional programming tools for control, including the necessary signal processing chain (digital at a minimu, analogue as well if some circuit design, that however we do not address herein, is present in their curriculum);
(6) simulate all feasible and meaningful compounds of the above;
(7) interpret simulation results as for
   (a) the aptitude of the devised control strategy to obtain the required objectives,
   (b) and the correspondence between the said strategy and the code to implement it.

4. THE PROPOSED SET OF TOOLS

The operations above, and many others, can be carried out (for example) with wxMaxima, Scilab, PIPE, OpenModelica and OpenPLC. All these tools are free software. We now briefly present them, to subsequently discuss their coordinated use and thereby motivate their choice.

4.1 The set components

wxMaxima was originally named MACSYMA for MAChine SYMbolic Algebra, which clearly indicates its purpose. Besides algebra *stricto sensu* it also offers calculus capabilities, including for example the Laplace transform and its inverse. Frankly speaking it may not be at the level of the major commercial symbolic tools, but for the needs of a control engineer it is more than enough.

Scilab is a mathematical package with a powerful scripting language, supported by a well established consortium. Should one prefer a more MATLAB-like tool, GNU Octave is a good community-maintained alternative.

PIPE stands for Platform Independent Petri net Editor, and allows both simulation and analysis (P- and T-invariants, liveness, boundedness and so forth).

OpenModelica is an industry-grade Modelica translator maintained by a solid consortium. It offers equation-based (both causal as block diagrams and a-causal) and algorithm-based modelling, including calling external C code, and allows to mix all approaches in a single model. It has graphical model editing and gives access to the comprehensive Modelica Standard Library. As models are written in terms of equations, also implicit, the tool provides a very natural way to write first-principle equations and get to the model directly.

OpenPLC is an environment composed of an editor for all five IEC 61131-3 languages, plus runtime modules for different platforms. There is a softPLC runtime to allow code execution on a PC, but also external hardware like the Arduino or the Raspberry Pi can be targeted.

As said we do not talk here about circuit design, but should this be needed a suitable tool could be KiCAd EDA (`https://www.kicad-pcb.org/`). The most straightforward way to distribute the set of tools to the students, in our opinion, is to create a virtual machine image. We are setting up one based on Ubuntu Linux, and shall make it available as soon as possible.

4.2 Coordinated components use and pedagogy

We refer to the list in Section 3, and for space reasons just scratch the surface. Extensions and generalisations are left to the reader's expertise, and to help the students we are complementing the mentioned virtual machine image with convenient examples.

Some items are straightforward. For example, (1a) is easily carried out with the wxMaxima commands `solve` to find equilibria, `jacobian` to express derivative matrices, `subst` to substitute equilibrium expressions, plus matrix manipulation ones; (1b) requires normal use of Scilab (Bode, Nyquist, hinf, lqg, and so on) while for block diagrams the Modelica.Blocks library contains all the necessary; (1c) can be done entirely with PIPE; (1d) is better executed on paper. The didactic outcome of such activities is that there are "elementary" operations that can be done with the various tools, that no tool does everything, that it is important to compose one's box of tools knowledgeably, looking at efficiency and not only at inter-tool communication because some way to take the output of a tool to the input of another can always be found and the effort spent in this will never cost as much as an inefficient tool—and last but not least, that paper and pen have citizenship in that box.

Items (2) through (4) have to do with understanding how a dynamic system is programmed—a core part of the envisaged pedagogy, as after years of experience and numerous control malfunctions just caused by not doing this programming properly, we reached the unwavering belief that "provided a control problem is small enough, you must be able of coming to the complete implementation of its solution with only calculus tools, a microcontroller, and a C compiler".

We have frequently heard the objection that teaching this is just wasting lecture time that should be better used for methodological subjects because then "code is generated", but from an educational perspective aimed to producing *methodologically* solid control *professionals* we strongly disagree. We provide below some motivations, ordered by increasing abstraction.

(1) Even if "a control engineer does not code", he/she may have to – and it would be educative – at the beginning of his/her career. Making him/her feel such an activity to have no intersection with previous university education is not a good idea.

(2) Being totally unaware of the IEC-to-code process hampers addressing high-performance applications; if skeptical, as a useful parallel, ask a high-end embedded systems developer if he/she could work e.g. with C without ever thinking about assembly code generation.

(3) Such a competence is necessary when part of a project is best realised in IEC textual languages like ST—and also to decide when this is the case, incidentally. More in general, it serves to select the most suited language for each project component; expecting a programmer without control culture to do that in the place of the control engineer is not a good idea either.

(4) The above competence is also often necessary for creating e.g. HIL simulators, as it may happen that for efficiency some parts of a control application will just need emulating with "equivalent code"—equivalent in the sense that from a system and control viewpoint it provides correct boundary conditions to the rest of the application, needless to say, not just that it has the same I/O structure.

(5) There are domains where code generation tools do not exist, so that one has to write controllers practically from scratch and obey to strict rules as for their integration with the controlled system; a strongly emerging such domain is quality of service control in cloud-based services.

(6) A control engineer with zero programming culture will not be able to lead programmers: the idea of preparing specifications and then just sending them out to coders without further interactions is simply utopia, and the said interactions require the control engineer to *also* understand speak the programmers' language. Exceptions are possible only in organisations that are large and strong enough to tolerate non negligible inefficiencies.

(7) For achieving the above envisaged culture about control software development, we cannot only rely on the computer basic courses that are typically present in a control curriculum. These teach programming in general and this is necessary, but for our needs there are too many things about software development that come from knowing control and not coding; these must be taught together with control, and by a somebody with strong control expertise.

(8) The objection could be extended to say that given the existence of symbolic tools, one does not need to know how to compute an integral. The answer "but that is theory" is inconsistent: *every* level of an engineering process has its own theory behind, including assembly generation. Of course one cannot know all, but must know enough to evaluate the outcome of an automated tool. If one has to know integrals for example to not just take the wxMaxima output for good acritically, the same should be true for code generation. If one knows about control software development, then he/she can use code generators safely: in the opposite case, we would not be equally confident.

Our experience is not exhaustive, of course, but along the years we have heard many former student complaints about a poor connection between control education and profession. What we would like to stress herein, based on the considerations made so far, is that a major reason – often somehow unconscious – for those complaints, is precisely the idea that we teach mostly theory to "not waste time with technology" and then the students will connect this education to professional training when needed. Most of them cannot make this connection alone; they need help to establish it right from the university. And this requires teaching to give technology a comparable dignity with respect to pure theory.

We end this digression by briefly discussing a subtle variation of the objection, that we should not waste time with technology "because it changes". This would be true

if we meant to teach this or that particular product, or family of products (recall the remark in Section 1 about "nuancing the question"). But what we propose here is different. We need to teach technological *principles*, and this is why we refer to *standards*. These are not so variable, and if well acquired, do help the student face this or that product and think that in the end if not for the position of buttons on screen they are the same. In the and, we could re-phrase the concept by saying that both theory and technology have their principles: from our educators' viewpoint, the issue is how to teach both in correlation with one another.

The above said, items (2–4) in the list require Scilab and/or C for coding and debugging, while wxMaxima can be used to obtain symbolically the lines of code related to computing the evolution of the state of the addressed dynamic system. For example, if the dynamic model

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \alpha x_1 + \beta x_2 + \gamma u \end{cases} \tag{1}$$

has to be simulated at (fixed) timestep q by using the implicit Euler discretisation technique, the lines of code necessary to compute the new state variables `(x1,x2)` based on their previous values `(x1old,x2old)` and on the input `u` are readily obtained in wxMaxima by issuing the commands

```
seq1  : dx1=x2;
seq2  : dx2=alpha*x1+beta*x2+gamma*u;
seqsd : subst([dx1=(x1-x1old)/q,dx2=(x2-x2old)/q],[seq1,seq2]);
upd   : solve(seqsd,[x1,x2]);
```

ready to copy and paste into the program editor. Strange as it may look, we encountered numerous problems that in fact simply originated from obtaining code for dynamic systems via error-prone manual manipulations instead of using the available tools. The didactic outcome here is consequently that such tools need integrating in the design process. Also and more practically, when one has to test a controller for correctness and therefore just needs to close the loop with some "credible" dynamics, obtaining crudely simplified "process" simulators as just done, and then programming them in the same environment as the controller, is often a simple and adequate solutions. Further lesson learnt, then, do not always take the full HIL platform if you just need to test a single control component: create a simple code for that, use it, document as needed, and store all into the project database.

Carrying on, items (5-7) in the list serve basically to put the students in contact with an industrial control development tool and have them understand that this is not "something else" (as too frequently heard) with respect to "those for theory". The students should see that the conclusions drawn on the model carry over to its realisation, and that for example a continuous-time simulation matches the C and the ST one. A particularly important role here can be played by OpenModelica, because in that environment it is possible to realise a modulating controller as continuous-time, discrete-time, complete algorithm and also external C code, and see the consistence of all realisations. It is also possible to realise logic controls in algorithmic form with any of the techniques taught to this end, and simulate the complete system including a representation of the process, generally in the continuous time for best realism.

It is finally important to educate the students, once they can walk through the entire control software development cycle with their box tool of tools, containing the most adapt to each purpose, not only to interpret simulations as per (7a–b) but also to decide which simulations to make, and to use these to possibly decide which tests shall be carried out on the real plant. Such decisions may have a significant impact on the timing and also on the economic aspects of a project, and from the professional viewpoint, having the students at least perceive the existence and main elements of this decision problem, is educationally invaluable. As for the collocation of the envisaged activities within the students' progression, the instructor willing to try the experience is surely in the position of judging. Some activities can apparently be done even at the introductory level, while others – the more related to industrial technology – are equally apparently advisable in subsequent courses. But this said, the choice has to be done based on the specific characteristics of the curriculum at hand.

5. RELATIONSHIPS WITH LABORATORIES

We spend some words on how the proposed box of tools can relate to existing laboratory equipment. In synthesis, and thinking mostly of advanced courses, our opinion is that if the equipment at hand is open enough as for interfacing with control hardware, then it can be used as the plant in the activities sketched above, or analogous ones at the instructor's choice. From this viewpoint, our box of tools has two relevant peculiarities. First, the presence of a Modelica tool makes it easy to construct first-principle models for the apparatuses. An example is provided in Figure 1, where the apparatus (Leva, 2003) is a metal plate heated by two transistors. The model is made of a-causal components and is multi-physics, as can be seen by distinguishing the electronic part (left) and the thermal one (right). Worth noticing is that all the used elements are taken from the Modelica Standard library.

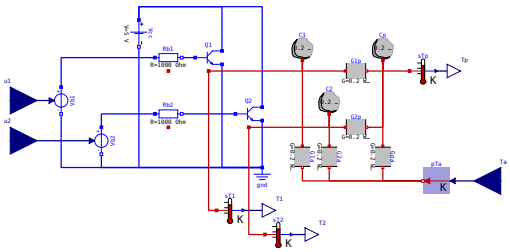

Figure 1. Modelica (a-causal, multi-physics) model of the experimental apparatus in Leva (2003).

Preparing such models is not difficult, and could also be carried out by the students themselves if some modelling and simulation course is involved in the teaching activity. Once models are available, the students can use them to test their control strategies; a (modulating only) example with a discrete-time LTI IMC controller for the above apparatus, to regulate the plate temperature with one transistor while the other provides a load disturbance, is shown in Figure 2.

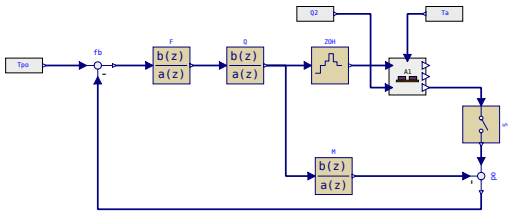

Figure 2. Modelica block diagram for the apparatus model of Figure 1 under discrete-time IMC control.

The students can verify their simulations on the real plant e.g. with OpenPLC runtime modules, thereby joining two advantages. First, the activity with the apparatuses is not confined anymore to just using pre-built control schemes. Second, the "custom" controls one decides to realise are implemented professionally, instead of just "coding" with no established industry standard in mind.

6. CONCLUSIONS AND FUTURE WORK

We discussed educating students to use software for implementing controls in a knowledgeable way, understand the involved standards (just mentioned for space limitations) to master the process of relating control software to models in a way that is significant from both the theoretical an the engineering standpoint. We believe this to be crucial for an effective professional use of the methodological competences acquired at the university. We substantiated our ideas in a set of tools to be employed in a coordinated manner, and connected to the content of the typical control courses. To facilitate trying out our ideas, we are preparing a virtual machine image that will be available as soon as possible. We are continuing the development of the concepts and the box of tools here described, also by creating *ad hoc* teaching material to include in the virtual machine under a Creative Commons licence. We hope that the presented work will be useful to the community and are open to any contribution, criticism, suggestion for improvement, and collaborations.

REFERENCES

Bellmunt, O., Miracle, D., Arellano, S., Sumper, A., and Andreu, A. (2006). A distance PLC programming course employing a remote laboratory based on a flexible manufacturing cell. *IEEE Transactions on Education*, 49(2), 278–284.

Bristol, E. (1986). An industrial point of view on control teaching and theory. *IEEE Control Systems Magazine*, 6(1), 24–27.

Candelas, F., García, G., Puente, S., Pomares, J., Jara, C., Pérez, J., Mira, D., and Torres, F. (2015). Experiences on using Arduino for laboratory experiments of automatic control and robotics. *IFAC-PapersOnLine*, 48(29), 105–110.

Coelho, A., Bruciapaglia, A., Simas, H., and Gomes, F. (1997). Low cost laboratory equipment for analysis and design of dynamic systems. *IFAC Proceedings Volumes*, 30(12), 99–104.

De Magalhães, A., Riera, B., and Vigário, B. (2011). When control education is the name of the game. In M. Cruz-Cunha, V. Vervalho, and P. Tavares (eds.), *Computer Games as Educational and Management Tools: Uses and Approaches*, 185–205. IGI Global, Hershey, PA, USA.

Dekemele, K., Chevalier, A., and Loccufier, M. (2018). ODYSC: A responsive educational web app for dynamics and control. *IFAC-PapersOnLine*, 51(4), 310–315.

González, I., Calderón, A., Mejías, A., and Andújar, J. (2016). Novel networked remote laboratory architecture for open connectivity based on PLC-OPC-LabVIEW-EJS integration. application in remote fuzzy control and sensors data acquisition. *Sensors*, 16(11), 1822–1845.

Gonzalez, J., Guzmán, J., Berenguel, M., and Dormido, S. (2012). A new framework to develop web-based interactive tools for control education. *IFAC Proceedings Volumes*, 45(11), 183–188.

Guzmán, J., Åström, K., Dormido, S., Hägglund, T., and Piguet, Y. (2006). Interactive learning modules for PID control. *IFAC Proceedings Volumes*, 39(6), 7–12.

Guzmán, J., Costa-Castello, R., Dormido, S., and Berenguel, M. (2016). An interactivity-based methodology to support control education: How to teach and learn using simple interactive tools [lecture notes]. *IEEE Control Systems*, 36(1), 63–76.

Heradio, R., de la Torre, L., Galan, D., Cabrerizo, F., Herrera-Viedma, E., and Dormido, S. (2016). Virtual and remote labs in education: A bibliometric analysis. *Computers & Education*, 98, 14–38.

Hoyo, A., Guzmán, J., Moreno, J., and Berenguel, M. (2015). Teaching control engineering concepts using open source tools on a Raspberry Pi board. *IFAC-PapersOnLine*, 48(29), 99–104.

Leva, A. (2003). A hands-on experimental laboratory for undergraduate courses in automatic control. *IEEE Transactions on Education*, 46(2), 263–272.

Leva, A. and Cimino, C. (2019). Teaching to design control applications with coordinated modulating and logic functions. In *Proc. 2019 European Control Conference*. Naples, Italy, 2019.

Maggio, M. and Leva, A. (2011). Teaching to write control code. *IFAC Proceedings Volumes*, 44(1), 7292–7297.

Montesinos Miracle, D., Galceran Arellano, S., Gomis Bellmunt, O., and Sudria Andreu, A. (2007). A new low-cost motion control educational equipment. In *Proc. 2007 European Conference on Power Electronics and Applications*, 1–6. Aalborg, Denmark.

Omar, H. (2018). Enhancing automatic control learning through Arduino-based projects. *European Journal of Engineering Education*, 43(5), 652–663.

Riera, B., Vigario, B., Chemla, J., Correia, L., and Gellot, F. (2009). 10 ans de maquettes virtuelles pour l'enseignement des automatismes: de WINSIM en 1998 à ITS PLC Professional Edition en 2008 (in French). *J3eA*, 8, 1004–1009.

Tekes, A., Van Der Horn, K., Marr, Z., and Tian, C. (2018). Dynamics, vibrations and control lab equipment design. In *Proc. ASME 2018 Dynamic Systems and Control Conference*, V002T16A001. Atlanta, GA, USA.

Vyatkin, V. (2013). Software engineering in industrial automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 9(3), 1234–1249.

Wenjiang, L., Nanping, D., and Tongshun, F. (2009). The application of Scilab/Scicos in the lecture of automatic control theory. In *Proc. 2009 IEEE International Workshop on Open-source Software for Scientific Computation*, 85–87. Guiyang, China.