An Introduction to Systems and Control Theory for Computer Scientists and Engineers

[Tutorial paper]

Alberto Leva
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci 32
20133 Milano, Italy
alberto.leva@polimi.it

ABSTRACT

This paper accompanies a tutorial aimed at introducing the basics of system and control theory so as to foster their utilisation for the management, but most important for the *design*, of computing systems. The tutorial is divided into three parts. The first one introduces the fundamental concepts of dynamic system and feedback and gives an overview of the properties that a control system has to enjoy, together with the main techniques to prescribe and assess these properties formally. The second part discusses a few computer-related application examples, revisiting the addressed problems from scratch with a system-centric viewpoint, and comparing the solutions - and most important, the way the system is viewed and designed - with state-of-the-art alternatives. This leads to envisage the potentialities of control-based computing systems design, but at the same time to identify open problems, both technological and methodological: an overview of these aspects is the subject of the third part. This companion paper motivates the tutorial, illustrates its *rationale*, and provides a commented outline.

CCS Concepts

•General and reference → Design; •Computing methodologies → *Modeling and simulation;* **•Software and its engineering** → *Software performance;*

Keywords

Systems and control theory, control-based computing system design.

ICPE'17, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3030207.3053677

1. WHY TEACH SYSTEMS AND CONTROL TO COMPUTER SCIENTISTS AND ENGINEERS

Control theory has been applied since a long time to manage computing systems for best performance [Abdelzaher et al. 2003]. This has given rise to a wealth of works on "computing systems control"—see e.g. the two books [Hellerstein et al. 2004] and [Janert 2013], spanning almost a decade, and their bibliographies.

The main *rationale* of computing systems control is to close feedback loops around a system, dynamically acting on some tuning parameters in such a way to optimise some measured metrics. Note that in doing so one implicitly assumes that in the absence of those control loops the system can operate, albeit not optimally. In a nutshell, therefore, the role of control theory in this setting is to provide a mathematical foundation to complement or replace heuristics in frameworks like the MAPE(-K) [IBM 2003].

The *scenario* just sketched is very well summarised in [Diao et al. 2005]. This paper, originating from an IBM research report, aims at exploring "the extent to which control theory can provide an architectural and analytic [notice the first adjective] foundation for building [notice the verb] self-managing systems" and showing the usefulness of establishing a "correspondence between the elements of autonomic systems and those in control systems", because "control theory provides a rich set of methodologies for building automated self-diagnosis and self-repair systems with properties such as stability, short settling times, and accurate regulation"—in one word, to ensure good and guaranteed performance.

If this viewpoint is taken, the main control skills required to use the control theory in computing systems are

1. model an existing and functional system, most frequently based on input/output measurements given the internal complexity of computing systems, and
2. design a control law to enforce the required properties.

Both these skills can be achieved with a quite basic and partial comprehension of the underlying theory – significantly more basic and partial than this proposal, to be explicit – and since the theory is well established and widely used in other domains, its application should be straightforward and successful.

As a literature and technology review too long to fit therein would reveal, however, things seldom go this way, and heuristics still plays a dominant role in the management of computing systems.

Heuristics is not evil by itself, of course, but if not properly confined, it can easily turn into "local problem by local problem" design, contrary to the systemic view that is typical of control theory. Sometimes, the complexity of systems designed by superimposing heuristic layers, reveals detrimental effects that a lack of systemic analysis allowed to stay concealed for a long time—a recent, impressing such story can be found in [Lozi et al. 2016]. Apparently, therefore, something is preventing control from fully unleashing its potential.

The main problem with the approach described so far is that *first* the system is designed, and *then* control moves in—this is another way to say that control loops are closed around *functional* systems, or from a slightly different standpoint, that design is not control-aware.

Indeed, if design is not carried out having control in mind, as is most frequently the case according to the story mentioned above and many others, or if the system is obtained by combining components designed previously with no awareness of their possible role in a control system, then a number of problems can – and do – arise: the available tuning knobs may be adequate for a human administrator sizing the system but not for controlling the required metrics online, components may reveal inadequate – if not unpredictably variable – timing properties, compounds can behave in ways unexpected by observing their components one by one, and so forth.

It has been recognised years ago that "software engineers need to use control concepts to master the ever-increasing complexity of computing systems" [Sanz and Årzen 2003, p. 14]. The author believes that this must also mean *designing* systems, or at least their critical parts, with control in mind, making the systems and control theory a means not only to manage complexity, but also and most important, to not create it unduly.

In this respect, recall the adjective "architectural" and the verb "building" evidenced in the quote from [Diao et al. 2005] above. Taking them in their full significance calls for much more than the mainstream "control of computing systems": it naturally leads to control-based computing system *design* [Leva et al. 2013]—an idea that has numerous declinations and can be applied to different depths, from instrumenting the system to make it easier to control with quite mainstream methods [Brun et al. 2009, Maggio et al. 2011, Patikirikorala et al. 2012], through the insertion of control-based specific active components [Hoffmann et al. 2013, Arcelli et al. 2015], down to a complete re-design of parts identified as critical [Terraneo et al. 2014, Al-Areqi et al. 2015].

Summing up, the author believes that the real potential of systems and control theory in the computing systems domain, is to formalise and set up management layers and loops, but also to help designing systems that are easily and well governed by those layers and loops.

In process control, the community has been talking for decades about "process/control co-design", see e.g. the historical paper [Ziegler and Nichols 1943], which is basically the same idea of building process and cotnrol together. Of course realising this in "industrial" domains in the strictest and somehow old sense of the term can be extremely difficult, because there the "process" can be something as heavy as a chemical plant. But in computers, where most objects to be controlled are software themselves, the idea of co-design should find a domain of election.

It has to be noted, however, that control-based design requires a deeper knowledge of systems and control than the basic one sketched above: in particular, it requires to master the subject not just as "yet another source of algorithms", but rather as a theoretical *corpus* to induce a *forma mentis* in analysing and designing the addressed systems. In the opinion of the author, for a proper use of control and an effective cooperation with control specialists when needed, computer scientists and engineers should be taught systems and control with the objective just outlined.

The problem is how so ambitious an objective can be achieved, and in the following a proposal is formulated. In detail, Section 2 talks basically about principles without necessarily thinking of a specific didactic form, while Section 3 specialises to the tutorial that this paper accompanies.

2. HOW TO TEACH (SOME IDEAS ON)

When teaching a subject new to a community, and that can affect their design habits significantly, quite often the best way to go is top down.

Specialising to the subject at hand, first one has to master the key ideas of *dynamic system* and *feedback* without thinking of any specific computer-related application. Otherwise, the temptation of classifying the presented control-based solutions against one's previous ideas – i.e., against a taxonomy that is inherently unfit to comprehend them – is irresistible, and if this temptation prevails, the pernicious idea of "control theory as a source of algorithms" is sowed.

On the contrary, first one has to be led to view dynamic systems as a formalism to describe the world, and perceive the generality of this description independently of its various flavours (continuous-time, discrete-time, event-based, and so forth); then, and in some sense as a consequence, one has to realise that feedback is already present in nature, simply as the way for systems to govern themselves and operate.

Once these two ideas are mastered, a conclusion quite naturally arises to guide the use of control in computers. In nature, systems start out with their feedback loops already installed—or, better, as a primary ingredient of their design. In computing systems one "creates one's physics", and wherever possible has to view feedback control as an ingredient of that physics, not a subsequent add-on.

More specifically, in computing systems one designs a "virtual physics" – think for example of the way jobs are created, accumulated and served in a queue network – employing resources from "real physics" like CPUs, disks, network transceivers, power supplies, and the like. The dynamic model of such a system entails unmodifiable parts – the laws of real physics – plus modifiable ones (referring again to the queue network case, all the queues', routers' and servers' management layers).

If the latter parts are designed in such a way to be well described by dynamic system, which is certainly and somehow naturally true for the former, then control moves in straightforwardly and successfully. If on the contrary the created physics is not keen to be modelled that way, for example because it was conceived directly as algorithms instead of dynamic systems generating algorithms, then control may

find so hard an obstacle that the only feasible solution is to close loops around the system *as is*—with the known and already mentioned limitations.

Indeed, and of course once again in the opinion of the author, teaching systems and control to computer people is primarily attaining the goals above. One may object that in so doing hardly any recipe to solve a problem is taught, but the author bears to state that attempting to proceed by recipes is exactly the reason why introducing control in the computer community seems so difficult (when seen from the control side, at least).

In fact, a computer scientist or engineer is not required to learn and apply the huge variety of control techniques – control people are there to do that – but rather and most important, to help create control-friendly systems (analogous, in some sense symmetrical considerations hold about teaching computer concepts to control people, but the matter is outside the scope of this paper).

Of course, once the cultural result just sketched out is gained, one can also illustrate some recipes for specific problems of if this is deemed convenient and interesting for the audience, but at this point the said specific applications cannot blur the mastered general ideas.

3. THE TUTORIAL – A MOTIVATED STRUCTURE OVERVIEW

As anticipated, the tutorial is divided into three main parts, that are briefly described in this final section.

The first part introduces dynamic systems as a mathematical object, defines their properties, and provides some engineering interpretations, avoiding the computer domain for the reasons outlined above in Section 2.

Then, the fundamental idea of feedback is introduced, concentrating on its role as a means to prescribe relevant properties like stability and performance, and some examples are worked out. Only at this point parallels are introduced with analogous ideas in computer-related works, such as the SASO properties [Hellerstein et al. 2004]. This first part of the tutorial hence provides the tools—but most important, the concepts and the viewpoint.

The second part works out a few design examples. The addressed problems are revisited from scratch, compatibly with the available time, so as to to show that the key point is not to apply another type of algorithm, but to change the design perspective.

In this part, the occasion is taken on one hand to structure the treatise distinguishing problem, model, solution and algorithm, and on the other hand also to organise the system design along the control-theoretical structuring into sensor, controller and actuator. Besides streamlining the matter, this facilitates future interactions of the audience with control people. In particular, for example, the focus is set on when a system is control-friendly and when it is not.

The second part also stresses the importance of system-level simulation, with synthetic models that not only provide the (model of) the controller, but at the same time can be suitable to assess formal properties without requiring too much detail on the system, nor information that should strictly pertain to downstream design phases.

The third and final part re-visits the previous two in a view to generalising and abstracting the proposed ideas, and foster a discussion on the audience concerning their interest, expectations, and impressions.

On one hand, the purpose is here to point out the potential benefits yielded by control-based computing systems design, as proven by the shown examples and as perceivable by the sketched problem characterisation in control terms. On the other hand, a number of open problems, from both the technological and the methodological standpoints. To date, systems appear far less control-friendly than they could and should be, and to fill this gap, both new theoretical elaborations and design practices are needed. The ultimate hope of the author is that the tutorial can give a contribution to the establishment of collaborations toward this important goal.

4. REFERENCES

[Abdelzaher et al. 2003] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu. 2003. Feedback performance control in software services. *IEEE Control Systems Magazine* 23, 3 (2003), 74–90.

[Al-Areqi et al. 2015] S. Al-Areqi, D. Görges, and S. Liu. 2015. Event-based networked control and scheduling codesign with guaranteed performance. *Automatica* 57, 7 (2015), 128–134.

[Arcelli et al. 2015] D. Arcelli, V. Cortellessa, A. Filieri, and A. Leva. 2015. Control theory for model-based performance-driven software adaptation. In *Proc. 11th International ACM SIGSOFT Conference on Quality of Software Architectures*. New York, NY, USA, 11–20.

[Brun et al. 2009] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. 2009. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee (Eds.). Springer, Berlin, Germany, 48–70.

[Diao et al. 2005] Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G.E. Kaiser, and D. Phung. 2005. A control theory foundation for self-managing computing systems. *IEEE journal on selected areas in communications* 23, 12 (2005), 2213–2222.

[Hellerstein et al. 2004] J. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. 2004. *Feedback control of computing systems*. John Wiley & Sons, New York, NY, USA.

[Hoffmann et al. 2013] H. Hoffmann, M. Maggio, M.D. Santambrogio, A. Leva, and A. Agarwal. 2013. A generalized software framework for accurate and efficient management of performance goals. In *Proc. 2013 International Conference on Embedded Software*. Montréal, Canada, Article No. 6658597.

[IBM 2003] IBM. 2003. An architectural blueprint for autonomic computing. *IBM White paper* (2003).

[Janert 2013] P.K. Janert. 2013. *Feedback control for computer systems*. O'Reilly Media, Sebastopol, CA, USA.

[Leva et al. 2013] A. Leva, M. Maggio, A.V. Papadopoulos, and F. Terraneo. 2013. *Control-based operating system design*. IET, London, UK.

[Lozi et al. 2016] J.P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, and A. Fedorova. 2016. The Linux scheduler: a decade of wasted cores. In *Proc. 11th European Conference on Computer Systems*. London, UK, 1–16.

[Maggio et al. 2011] M. Maggio, H. Hoffmann, M.D. Santambrogio, A. Agarwal, and A. Leva. 2011. Decision making in autonomic computing systems: Comparison of approaches and techniques. In *Proc. 8th ACM International Conference on Autonomic Computing*. Karlsruhe, Germany, 201–204.

[Patikirikorala et al. 2012] T. Patikirikorala, A. Colman, J. Han, and L. Wang. 2012. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proc. 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. Zürich, Switzerland, 33–42.

[Sanz and Årzen 2003] R. Sanz and K.E. Årzen. 2003. Trends in software and control. *IEEE Control Systems Magazine* 23, 3 (2003), 12–15.

[Terraneo et al. 2014] F. Terraneo, L. Rinaldi, M. Maggio, A.V. Papadopoulos, and A. Leva. 2014. FLOPSYNC-2: sub-microsecond, sub-μA clock synchronisation for wireless sensor networks. In *Proc. IEEE Real-Time Systems Symposium RTSS 2014*. Roma, Italy, 11–20.

[Ziegler and Nichols 1943] J.G. Ziegler and N.B. Nichols. 1943. Process lags in automatic control circuits. *Transactions of the ASME* 65, 5 (1943), 433–443.