Design of an Embedded Control Systems Laboratory Experiment

Pau Martí, *Member, IEEE,* Manel Velasco, Josep M. Fuertes, *Member, IEEE,* Antonio Camacho, and Giogio Buttazzo *Senior Member, IEEE*

***Abstract*—This paper presents a prototype laboratory experiment to be integrated in the education of embedded control systems engineers. The experiment, a real-time control of a dynamical system, is designed to drive students to a deeper understanding and integration of the diverse theoretical concepts that often come from different disciplines such as real-time systems and control systems. Rather than proposing the experiment for a particular course within an embedded systems engineering curriculum, the paper describes how the experiment can be tailored to the needs and diverse background of both undergraduate and graduate students education.**

***Index Terms*—Embedded systems education, real-time systems, control systems, laboratory experiment.**

I. Introduction

THE economic importance of embedded systems has grown exponentially as electronic components are in everyday use devices. Hence, embedded systems education is a strategic asset, and university curricula are being adapted accordingly to cover this domain [1]. Embedded systems courses are being integrated into existing science and engineering curricula [2], [3], but also specific curricula have been developed to integrate the broad set of concepts into a course sequence [4]–[7]. In addition, modern teaching practices, such as problem based learning [8], international project collaboration [9], cooperative learning [10], on-line competitions [11], educational games [12], or remote laboratories [13], have also been applied to the embedded system education. To provide students with in-depth understanding across all the areas and disciplines involved in embedded systems is a difficult task. Hence, laboratory activities are crucial to consolidate the diverse theoretical material [14].

Since many embedded systems are control systems [15], and considering that there is an increasing trend to adopt real-time technology for the embedded computing platform [16], laboratory experiments including topics of real-time and control systems are becoming more and more important for the education of embedded systems engineers. The traditional teaching approach to real-time systems and to control systems courses can be quite math-intensive and abstract, thus failing to introduce students to the realities of embedded control system implementation. Moreover, the natural interaction and integration between these two disciplines is often neglected due to the traditional compartmentalized nature of science and engineering education. Since control systems are real-time systems, control engineers must have an understanding of computers and real-time systems, while computer engineers must understand control theory.

To overcome such limitations, this paper presents a laboratory experiment to be integrated in the education of embedded control systems engineers that flexibly combines two main disciplines: real-time systems and control systems. The flexibility is achieved by describing a set of problems/observations that provide the spectrum of possible choices that instructors/students have and the work that has to be done to complete the experiment. This permits elaborating diverse assignments for the laboratory experiment with open problems rather than providing tight guidelines, while providing the tools for assessing whether the students design and implementation choices were correct. Finally, through the experiment, it is shown that the combination of both disciplines do not rise conflicts. It rather provides complementary approaches/views that help in the multidisciplinary learning process required in the embedded systems education.

The experiment main activity includes the implementation of a real-time control application, consisting in controlling a physical plant by a controller implemented as a software task executing on top of a real-time operating system (RTOS). Rather than proposing an experiment for a particular course within an embedded systems engineering curriculum, the paper describes how the experiment can be tailored to the needs of both undergraduate and graduate students education, and to the diverse background of the target audience. A tentative laboratory program covering the different stages required to carry out the experiment is presented, and its integration into a master-level students course is also reported.

The rest of the paper is organized as follows. Section II sets the objectives, competence and learning outcomes for the designed experiment. Section III discusses the selection of the controlled plant and processing platform/RTOS. Section IV introduces the set of problems/observations for carrying out the activity. Section V presents an outline of the experiment and its adaptation to a course. Section VI concludes the paper.

Manuscript received March 10, 2009; revised June 15, 2009; accepted for publication December 23, 2009. This work was partially supported by NoE ArtistDesign IST-FP7-2008-214373, and by Spanish Ministerio de Educación y Ciencia Project ref. CICYT DPI2007-61527.

P. Martí, M. Velasco, Josep M. Fuertes and Antonio Camacho are with the Automatic Control Department, Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, Spain. E-mail: {pau.marti, manel.velasco, josep.m.fuertes, antonio.camacho.santiago}@upc.edu

G. Buttazzo is with The Computer Engineering Department, Scuola Superiore Sant'Anna of Pisa, RETIS Lab, Area CNR Via Moruzzi, 1, 56124 Pisa, Italy. E-mail:giorgio@sssup.it

TABLE I
REQUIRED SKILLS

1	Understanding embedded control systems, their importance, limitations, restrictions, and application areas
2	Analysis and synthesis capabilities
3	Explanation capability for efficient oral and written communication
4	Capability of integrating autonomous learning with team work
5	Capability of analyzing and assessing the economical, social and environmental impact of the solutions

TABLE II
LEARNING OUTCOMES

1	Identify embedded systems features
2	Identify components, concepts and design methodologies
3	Interpret data-sheets, documentation and specifications
4	Design, build and troubleshoot an embedded control system
5	Practice on modelling, analysis and design of control systems
6	Practice on real-time programming and operating systems
7	Evaluate system performance

II. OBJECTIVES, COMPETENCE AND LEARNING OUTCOMES

The experiment objectives are twofold. First, the positive benefits of experimental learning are well known in education and professional activities. Students confidence and enthusiasm in experiments grow as they practice in problem solving, team work, design skills, etc. Second, and more specifically, experiments should educate students in embedded control systems, providing additional knowledge they cannot acquire from theory.

Looking at the ECTS (European Credit Transfer System), program objectives are preferably specified in terms of the learning outcomes and competence to be acquired. Although the proposed experiment is not tailored to a specific program, nor to any specific level of study (undergraduate, graduate), the main skills to be acquired by students are listed in Table I. Skills 1-2 relate to technical aspects and theoretical knowledge on embedded control systems, skills 3-4 relate to practical issues and experimental learning, whereas skill 5 refers to sustainability issues. The multidisciplinary nature of embedded systems requires more background and transversal knowledge in different fields, combined with the capability of integrating different skills for a system wide objective. The learning outcomes are more specific because they state what is expected from a student as a result of the learning process.

The learning outcomes are listed in Table II. The first three outcomes are related to understanding embedded control systems, from a technical point of view, taking into account the multidisciplinary nature of the field. In this process, it is also crucial for students to be able to read, understand, and use existing documentation, like data-sheets, application programming interface (API) reference manuals, etc. Outcomes 4 to 6 are related to implementation issues, which are essential for reproducing an experiment. Finally, the evaluation of system performance is crucial to assess if the specifications are met.

The required background for students to carry out the experiments is a basic knowledge on control systems theory and real-time programming using an operating system.

III. SELECTION OF THE CONTROLLED PLANT AND PROCESSING PLATFORM

The controlled plant and processing platform (hardware and real-time operating system) have been carefully selected to have a friendly, flexible, and powerful experimental set-up. Both must be simple to avoid discouraging those students with strong control systems background but weak real-time systems background when facing the programming part, or vice-versa, to avoid discouraging those students with strong real-time systems background but weak control systems background when facing the controller design stage.

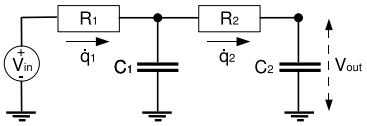
A. Plant

Many standard basic and advanced controller design methods rely on the accuracy of the plant mathematical model. The more accurate the model, the more realistic the simulations, and the better the observation of the effects of the controller on the plant. Hence, the plant was selected among those for which an accurate mathematical model could easily be derived.

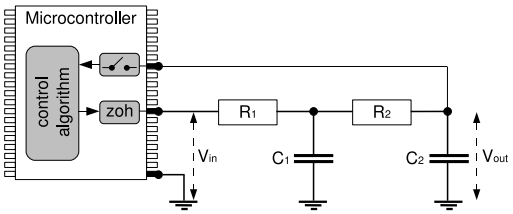
Plants such as an inverted pendulum or a direct current motor are the defacto plants for benchmark problems in control engineering [17]. However, their modelling is not trivial and the resulting model is often not accurate. This leads to a first controller design that has to be adjusted by ”engineering experience”, thus requiring knowledge that is difficult to formalize and transmit to the students. To avoid such a kind of drawbacks, a simple electronic circuit in the form of an $RCRC$ (Figure 1a) was selected. The simplicity of its components and their simple and intuitive physical behavior have been the main reasons for its selection. Note however that experiments using other plants can be complementary to the approach presented here, e.g. [15], [18]–[20]. Indeed, Lim [19] also proposes electronic circuits. However, they are slightly more sophisticated because they include operational amplifiers. Although richer dynamics can be achieved, the intuitive behavior and thus the modeling of operational amplifiers is not straightforward.

The selection of an electronic circuit as a plant has also another important advantage: depending on the specific circuit, it can be directly plugged into a micro-controller without using intermediate electronic components, as shown in Figure 1b, where the *zoh* box (zero order hold) represents the actuator and the box above represents the sampler. That is, the transistor-transistor logic (TTL) level signals provided by the micro-controller can be enough to carry out the control. Note that this is not the case, for example, for many mechanical systems. Such a simplification in terms of hardware reduces the modelling effort to study the plant and no models for actuators or sensors are required. Additional benefits of these types of plants are that systems can be easily built, are cheap, have light weight, and can be easily transported and powered.

The control objective would be to have the circuit output voltage V_{out} (controlled variable) to track a reference signal or to settle to a constant value while meeting for example given transient response specifications, which mandates to use tracking structures. The control will be achieved by sampling V_{out}, executing the control algorithm and applying the calculated



(a) RCRC circuit



(b) Control setup

Fig. 1. Plant and control setup.

control signal to the circuit via varying the circuit input voltage V_{in} (manipulated variable). Disturbances can be injected by a variable load voltage placed in parallel to the output voltage.

B. Processing platform

The processing platform consists of the hardware platform and the real-time operating system. As hardware platform, a micro-controller based architecture was selected because embedded systems are typically implemented using this type of hardware. Note, however, that too small micro-controllers may not be powerful enough for running an RTOS, as discussed in [22]. Among the several possibilities available on the market [23], it was decided to adopt the Flex board [24].

The Flex board (in its full version) represents a good compromise between cost, processing power, and programming flexibility. It was produced as a development board for building and testing real-time applications using standard components and open source software. The board includes a Microchip dsPIC DSC micro-controller dsPIC33FJ256MC710, a socket for the 100 pin Plug-In Module (PIM), an ICD2 (in-circuit debugger) programmer connector, a USB (Universal Serial Bus) connector for direct programming, power supply connectors, a set of leds for monitoring the board, an on-board Microchip PIC18F2550 micro-controller for integrated programming, and a set of connectors for daughter boards piggybacking.

The board has several key benefits that make it suitable to be used for educational purposes. First has a robust electronic design, which is an important feature when it is employed by non skilled users. Second, it has a modular architecture, which allows users to easily develop home-made daughter boards using standards components. A set of daughter boards can be added to the Flex board for easy development, such as a multi-bus board equipped with CAN (Controller Area Network), Ethernet, I2C (Inter-Integrated Circuit), and other communication protocols. On one hand, the availability of CAN or Ethernet permits to build and experiment with networked control applications. On the other hand, the available networks can be used for debugging purposes or for extracting data from the board, which is a difficult task when dealing with embedded systems.

As far as the real-time kernel is concerned, different possibilities were considered. First, many well-known real-time operating systems, such as real-time Linux [25], target processors that may be too powerful for embedded applications. But more important, their internal structure is often too complex for those students with a low profile in (real-time) operating systems. Hence, it looks more desirable to work with small real-time kernels (see e.g. [26]–[31] for small real-time kernels targeting small architectures) whose internals are accessible, easy to understand and modify, in order to tailor them to the specific application needs. On the other hand, from a user point of view, programming and configuring the kernel (including creating tasks, assigning priorities/periods/deadlines, and setting the scheduling policy) should be friendly enough to attract non-skilled programmers.

From the considerations mentioned above, Erika Enterprise real-time kernel [24] was selected. Erika provides full support to the Flex board in terms of drivers, libraries, programming facilities, and sample applications. The kernel, available under the General Public License and OSEK (Open Systems and their Interfaces for the Electronics in Motor Vehicles, [32]) compliant, is a RTOS for small micro-controllers based on an API similar to those proposed by the OSEK consortium. The kernel gives support for preemptive and non-preemptive multitasking, and implements several scheduling algorithms [33]. The API provides support for tasks, events, alarms, resources, application modes, semaphores, and error handling. All these features permits to enforce real-time constraints to application tasks to show students the effects of sampling periods, delays and jitter on control performance.

The development environment for Erika Enterprise is based on cross-compilation, avoiding typical students misconceptions when the development platform and the target share the same hardware. A tool, named RT-Druid [24] (based on Eclipse [34]), can be used as a default development platform to program in C, with support from Microchip for the compiler and for the programming development kit. The latter is important because Microchip web-pages [35] are always a good place where to share experiments experiences and code: a good place for instructors and students to visit. RT-Druid implements an OIL (OSEK Implementation Language) language compiler, which is able to generate the kernel configuration from an OIL specification. Apart from programming in C, the Flex board can also be programmed automatically using the Scilab/Scicos [36] code generator (similar to what can be done with MATLAB/Simulink [37] and its Real-Time Workshop, as used for example in [38] for rapid control prototyping). This is an important benefit for non-skilled C programmers.

From an education point of view, it is also important to note that there is the possibility to build a community around this processing platform to create a repository of control software for education. In fact, a set of *application notes* that describe a set of control experiments (inverted pendulum, ball and plate, etc.) developed with Erika on Flex can be found in [24].

Finally, it must be stressed that the price of the Flex board lies in the lower bound of evaluation board prices, and that

the Erika kernel and the associated development tools are open source, available for free, or available for free in student edition format. Hence, it is an economically atractive option.

IV. Example of the Lab Experiment

This section presents some of the activities in the form of problems and solutions (and observations) required to carry out the lab experience, which are later ordered in the work plan. The emphasis is in the control analysis and design part.

A. Problem 1. Plant modelling

If q_i represents the charge on capacitor C_i, the differential equations of the circuit, in terms of the currents $\dot{q}_i$ at each R_i, are given by

$$\begin{aligned} \dot{q}_1 R_1 + (q_1 - q_2)\frac{1}{C_1} &= V_{in} \\ \dot{q}_2 R_2 + (q_2 - q_1)\frac{1}{C_1} + q_2\frac{1}{C_2} &= 0 \\ q_2\frac{1}{C_2} &= V_{out}, \end{aligned} \tag{1}$$

For example, using state-space formalism, a state-space form is given by

$$\begin{aligned} \dot{x}(t) = & \begin{bmatrix} 0 & 1 \\ \frac{-1}{R_1R_2C_1C_2} & -\frac{R_1C_1+R_2C_2+R_1C_2}{R_1R_2C_1C_2} \end{bmatrix} x(t) \\ & + \begin{bmatrix} 0 \\ \frac{1}{R_1R_2C_1C_2} \end{bmatrix} u(t) \\ y(t) = & \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) \end{aligned} \tag{2}$$

where $u(t)$ is the control signal, $y(t)$ is the plant output, and $x(t) = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ is the state vector, where x_1 corresponds to the output voltage V_{out}, and x_2 is $\dot{q}_2/C_2$.

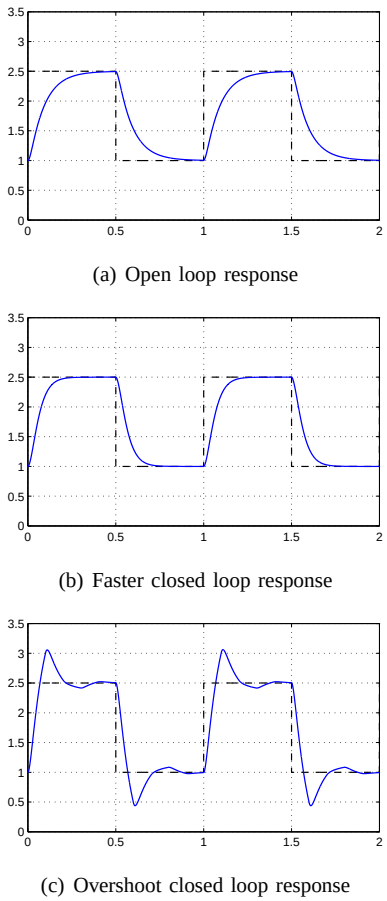
Observation 1: The modelling of the plant could have been also done in terms of a transfer function (see [39] for the analysis). Even obtaining the differential equations is a good exercise. Adopting the state space formalism may add another benefit if using model (2). Since only the output voltage x_1 can be physically measured, the control algorithm requires the use of observers for predicting x_2. This opens the door to experiment with several types of observers and the implementation of the controller has to include them. Also, the selection of the state variables is arbitrary, and therefore, students have to take design decisions. For example, the voltages in both capacitors could also have been chosen as a state variables. In any case, if possible, it is interesting to chose the state variables in such a way that the controlled variable is directly available through the output matrix in order to minimize computations in the micro-controller.

B. Problem 2. Electronic components

The selection of the electronic components is very important for several reasons. The output impedance must be low enough to properly connect the circuit to the analog-to-digital converter (ADC) or to some external instrumentation, such as an oscilloscope. For example, given the initial components $R_1 = R_2 = 1$ KΩ and $C_1 = C_2 = 33$ μF, a manageable circuit impedance is obtained. With these components, the state space model becomes

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ -976.56 & -93.75 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 976.56 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} x(t). \end{aligned} \tag{3}$$

Observation 2: Students can be given other values. For example, with $R_1 = R_2 = 330$ KΩ and $C_1 = C_2 = 100$ ηF, it is easy to see that the equivalent output impedance is too high for the ADC. To derive such a conclusion, students have to consult the dsPIC data sheet.



(a) Open loop response

(b) Faster closed loop response

(c) Overshoot closed loop response

Fig. 2. Simulated RCRC responses.

C. Problem 3. Open loop simulation

Open loop dynamics can be observed by injecting reference signals to the circuit via its input V_{in}. For example, by injecting a square wave that oscillates between 1 V and 2.5 V at 1Hz, the obtained dynamics are illustrated in Figure 2a. The voltage output V_{out} (solid curve) slowly tracks the reference (dashed curve).

Observation 3: The electronic components determine the circuit open loop dynamics. Students must be aware of this by playing with different electronic components. In addition, simulation can be done by standard software packages used in control engineering (e.g., MATLAB/Simulink, Scilab/Scicos)

or by programming the response in C or even using an spreadsheet application.

D. Problem 4. Controller design: sampling period and performance specifications

For the chosen plant, there can be different control goals. A possibility can be to modify the $RCRC$ transient response by accelerating it, or by arbitrarily achieving a given overshoot. In any case, the selection of the sampling period and the controller itself have a strong impact on control performance. Also, they have to be selected and designed taking into account the processing platform. If avoiding intermediate electronics between plant and dsPIC is the assumption, the control signal (V_{in}) can be generated using the Pulse Width Modulation (PWM) (by adjusting the duty cycle) and the controlled variable (V_{out}) can be obtained through the ADC. Therefore, the reference, the sampling period, and the controller must be chosen/designed so that the voltage levels of the control signal and generated peak current levels lie within the hardware limitations.

Given the previous square reference signal, through an iterative design stage, and according to standard rules of thumb, two state feedback controllers can be designed, one that accelerates the response, named *fast* controller, and another that produces overshoot, named *overshoot* controller. For the fast controller, the period is set to $h = 0.01$ s, and the discrete state feedback controller is designed to place the continuous closed loop poles at $p_{1,2} = -30$ (note that the open loop system has poles at -12 and -81, approximately). The overshoot controller has a sampling period of $h = 0.1$ s, and the controller places the closed-loop poles at $p_{1,2} = -10 \pm 20i$.

It is worth noting that, using only standard rules of thumb for selecting sampling periods [39], both controllers should have a slightly shorter sampling period. The *fast* controller should be given a period of $h = 0.007$ s and the *overshoot* controller a period of $h = 0.03$ s. However, since longer sampling periods result in lower controller resource demands, an iterative simulation process was used to find longer sampling periods for both controllers without introducing unacceptable performance degradation.

Observation 4: Selecting sampling periods, desired closed loop poles, or even having a higher amplitude for the reference signal may cause the values of the control signal to be out of range. Relations illustrating trade offs in the design of embedded control systems can also be taught to the students, such as showing that increasing sampling rates means stronger control signals (saturation problem) but also more processor usage (feasibility/schedulability problem).

E. Problem 5. Controller design: tracking

The controller design has to consider that the goal is to track a voltage. The standard tracking structure [39], that includes N_u as the matrix for the feed-forward signal to eliminate steady-state errors and N_x as the matrix that transforms the reference r into a reference state, can be adopted. Following the case study, $N_u = [1]$ and $N_x = [1 \quad 0]$, and discrete gains are $K = [0.0685 \quad -0.0249]$ or $K = [1.0691 \quad -0.0189]$ for the fast or overshoot controller.

```
CPU mySystem {
 OS myOs {
  EE_OPT = "DEBUG";
  CPU_DATA = PIC30 { APP_SRC = "code.c";MULTI_STACK = FALSE;
                     ICD2 = TRUE;};
  MCU_DATA = PIC30 { MODEL = PIC33FJ256MC710;};
  BOARD_DATA = EE_FLEX { USELEDS = TRUE;};
  KERNEL_TYPE = EDF { NESTED_IRQ = TRUE;TICK_TIME = "25ns";};
  };
  TASK myTask {
   REL_DEADLINE = "10ms";PRIORITY = 1;STACK = SHARED;
   SCHEDULE = FULL;
  };
  COUNTER myCounter;
  ALARM myAlarm {
   COUNTER = "myCounter";
   ACTION = ACTIVATETASK {TASK = "myTask";};
  };
};
```

Fig. 3. Kernel configuration file

Observation 5: Students can practice other tracking structures, such as integral control, and can study whether the code of the controller would suffer significant changes.

F. Problem 6. Controller design: closed loop simulation

The simulated closed loop response for the fast and overshoot controllers is shown in Figure 2 b) and c), respectively.

Observation 6: As before, simulations can be done using different methods.

G. Problem 7. Controller design: observers

For the simulation, the two state variables are available. However, in the real experiment, an observer must be included. For simplicity in coding the control task, a reduced observer can been chosen for observing the second state variable x_2. For example, the observer discrete gain is $K_r = -37.81$ or $K_r = -13.42$ for the fast and overshoot controller if the observer continuous closed loop pole are located at $p_{ob} = -50$.

Observation 7: Students can design and evaluate by simulation different types of observers (reduced, complete, etc.) with different dynamics. That is, they can also evaluate the effect of different locations for the observer poles. From an implementation point of view, students can also assess the effect that splitting the control algorithm into two parts (calculate control signal and update state) has on input-output delays and schedulability [41].

H. Problem 8. Implementation: kernel configuration and control algorithm

The first implementation involves coding the controller in a periodic task that will execute in isolation on top of Erika. The main pseudo-codes are illustrated in Figures 3, 4 and 5.

Figure 3 is the *conf.oil* file that specifies the kernel configuration with EDF (Earliest Deadline First, [40]) scheduling algorithm and a periodic task that will be used to implement, for instance, the fast controller. The basics of the main code are illustrated in Figure 4. First, the timer T1 is initialized and, together with *SetRelAlarm*, will produce the periodic activation

```
int main(void)
{
 T1_program();
 EE_time_init();    // EDF initialization
 ADC_init();        // ADC1 configuration
 PWM_config();      // PWM1 configuration
 SetRelAlarm(myAlarm, 1, 10);
 for (;;)           // reference signal
 {                  // generation
   Delay(100000);
   reference = 2.5;
   Delay(100000);
   reference = 1;
 }
 return 0;
}
```

Fig. 4. Main code

```
TASK(myTask)
{
  x_1 = read_adc()
  r   = reference;
  x_2 = observer(Kr,x1,r,x_1old,x_2old);
  u   = r*NU + K_1*(r*NX_1 - x_1) + k_2*(r*NX_2 - x_2);
  write_pwm(u);
  x_1old = x_1;
  x_2old = x_2;
}
```

Fig. 5. Control task code

of the control task every 10 ms (the processor speed was configured at 40 MIPS - Million Instructions Per Second). Figure 5 shows the control task code including the observer and the tracking structure.

Observation 8: Carrying out the kernel configuration and programming the controller could be taught following an ordered sequence of steps, such as: 1) introduction to kernel configuration, 2) introduction to periodic tasks, 3) introduction to input/output operations (PWM, ADC).

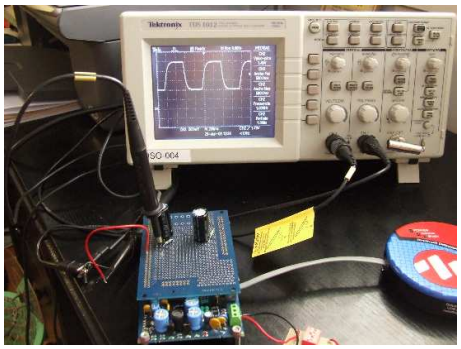
I. Problem 9. Implementation: setup and monitoring details

Figure 6a shows the experimental setup that includes an oscilloscope (for displaying the circuit output voltage) to show the open loop response (Figure 6b) and the closed loop responses (Figures 7a and b) achieved by each controller executing in isolation. The oscilloscope screen-shots confirm that the implementation achieves the control goal: the system output performs the desired fast tracking or achieves the specified overshoot.

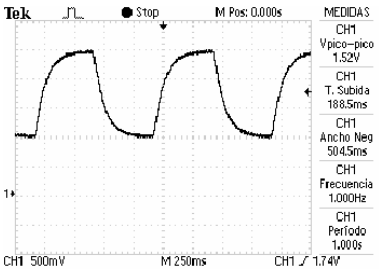
Observation 9: An oscilloscope has been used to monitor both the responses. It can also be of interest to monitor whether the control task executes when specified. Another option could be to use the Multibus board to send the data of interest via Ethernet or any other available communication protocol. This would pose interesting challenges in terms of the real-time system such as non-invasive debugging.

J. Problem 10. Multitasking: simulation

A second implementation is introduced to illustrate more advanced concepts. In the previous implementation a control task was executing in isolation. However, in many high-tech systems, the processor is used not only for the control

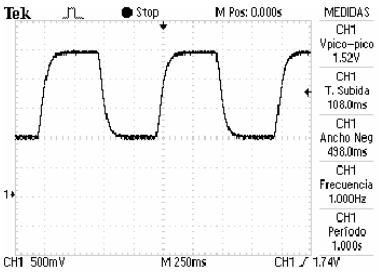


(a) Setup

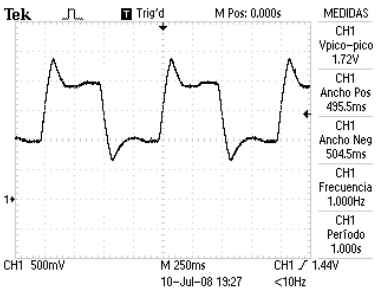


(b) Open loop response

Fig. 6. Experiment setup and monitoring.



(a) Fast closed loop response

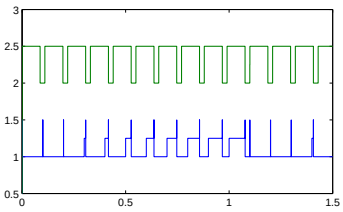


(b) Overshoot closed loop response

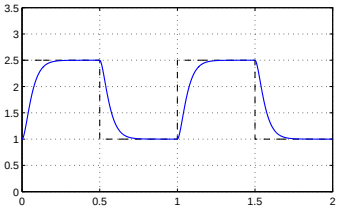
Fig. 7. RCRC responses.

computation, but also for interrupt handling, error management, monitoring, etc. And it is known that in a multitasking real-time control systems, jitters, i.e. timing interferences on control tasks due to the concurrent execution of other tasks, deteriorate control loops performance [41]. The objective of this implementation is to observe these degrading effects and implement corrective actions, e.g. [42]–[44].

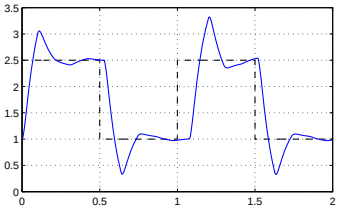
A starting point is to inject a new task in the kernel for each control task. The new task, named noisy task, when to



(a) Partial schedule

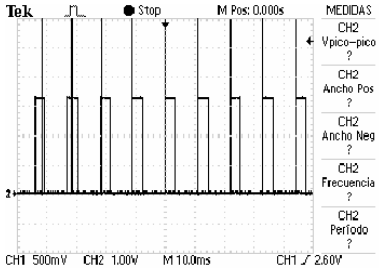


(b) Faster multitasking closed loop response

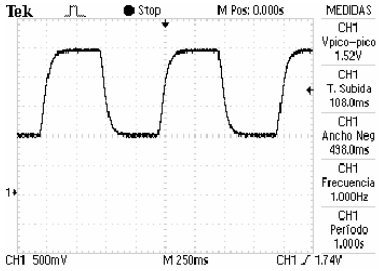


(c) Overshoot multitasking closed loop response

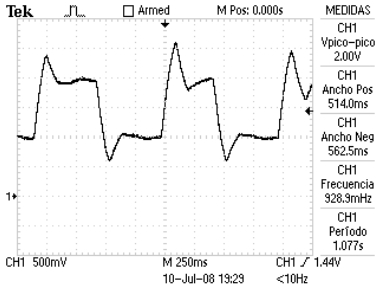
Fig. 8. Simulated multitasking RCRC responses and schedule.



(a) Partial schedule



(b) Faster multitasking response



(c) Overshoot multitasking response

Fig. 9. Implemented multitasking RCRC responses and schedule.

be executed together with the fast controller, is given a period (and relative deadline) of 11 ms, and it is imposed an *artificial* execution time of 9 ms. The noisy task that goes together with the overshoot controller has a period and relative deadline of 110 ms with 90 ms of execution time. The simulation of each multitasking system was done in the TrueTime simulator [45]. Putting together each control task with the corresponding noisy task under EDF results in timing variability (jitter) for the control task, as illustrated for the first multitasking system in the schedule of Figure 8a. In this figure, the bottom curve represents the execution of the fast controller task, whereas the top curve represents the execution of the noisy task. In each graph, the low-level line denotes no-execution (that is, intervals in which the processor is idle), the middle level line denotes a task ready to execute (i.e., waiting in the ready queue), whereas the high-level line denotes a task in execution. Note that the control task has a measured execution time of 0.12 ms, which is much less than the noisy task.

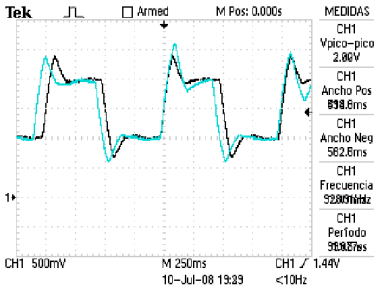
Looking at the plant responses in Figures 8 b) and c), it can be appreciated that the fast controller does not exhibit a control performance degradation, while the overshoot controller suffers some degradation: overshoots are bigger, square amplitude differs, transient response varies, etc. This fact indicates that the current control design for the fast controller is robust against jitters induced by scheduling, while the second one is more fragile.

Observation 10: Adopting an iterative simulation study, students can learn which parameters play an important role when jitters appear. Are shorter sampling periods, or non-overshootted responses, a guarantee for having robust control designs? Which role do deadlines play in reducing jitters? For further questions and solutions, see [46] and references therein.

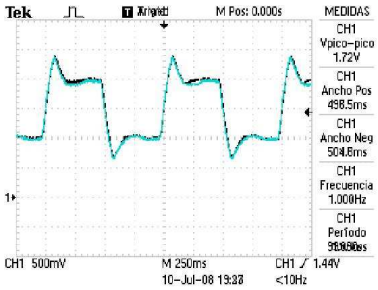
K. Problem 11. Multitasking: implementation and monitoring details

The new implementation requires specifying the noisy task by modifying the kernel *oil* file in terms of defining the new task and the associated alarm. Also, the new task has to be coded: forcing an artificial execution time is achieved by placing a delay into the code. The *main* code has to be modified to configure the new alarm associated to the new task.

After the implementation, in the first multitasking system, it can be verified that scheduling conflicts (as illustrated in the simulated schedule shown in Figure 8a) may occur, as illustrated in Figure 9a. In this sub-figure, the execution of the fast controller task sets an output pin to 0 and 1 at each job start and finishing time. And the noisy task sets an output pin to 0 and 0.5 at each job start and finishing time. In addition, similar responses for the fast and overshoot controller are obtained (see sub-figures 9 b) and c)), showing that the overshoot controller suffers degradation from jitters,



(a) Overshoot controller with/without jitters



(b) Overshoot controller eliminating the jitter problem

Fig. 10. Overshoot controller: degradation and solution

while the fast controller shows the same response as it is executed in isolation. To illustrative purposes, Figure 10a shows the overshoot controller response when executing in isolation (dark curve) and when executing in the multitasking system (grey curve).

Observation 11: Undergraduate students may work the experiment up to this problem. It shows the importance of concurrency and resource sharing with respect to control performance in a multitasking embedded control system.

L. Problem 12: Multitasking: design for eliminating or minimizing the jitter problem

Recent research literature has faced the problems introduced by jitter and many solutions have been proposed. Here, the solution proposed by Lozoya et al. [44] has been adopted.

The basic idea is to synchronize the operations within each control loop at the actuation instants. In this way, the time elapsed between consecutive actuation instants, named t_{k-1} and t_k, is exactly equal to the sampling period, h. Within this time interval, the system state is sampled, named $x_{s,k}$, and the sampling time recorded, $t_{s,k} \in (t_{k-1}, t_k)$. The difference between this time and the next actuation time

$$\tau_k = t_k - t_{s,k} \tag{4}$$

is used to estimate the state at the actuation instant as

$$\hat{x}_k = \Phi(\tau_k)x_{s,k} + \Gamma(\tau_k)u_{k-1} \tag{5}$$

where $\Phi(t) = e^{At}$ and $\Gamma(t) = \int_0^t e^{As}\mathrm{d}sB$, being A and B the system and input matrices in (3), and u_{k-1} the previous control signal. Then, making use of $\hat{x}_k$, the control command is computed using the original control gain K as

$$u_k = K\hat{x}_k. \tag{6}$$

The control command u_k is held until the next actuation instant. A control strategy using (4)-(6) relies on the time reference given by the actuation instants, if u_k is applied to the plant by hardware interrupts, for example. In addition, samples are not required to be periodic because τ_k in (4) can vary at each closed-loop operation.

After implementing this strategy on the overshoot controller in the multitasking system, Figure10 b) shows the result. Specifically, it shows the overshoot controller response when executing in isolation (dark curve) and when executing in the multitasking system using the algorithm that eliminates jitters (grey curve).

Observation 12: The problem presented above and its solution can be split into several tasks, like analysis and modelling of the new control algorithm, implementation of the control algorithm, etc. An interesting issue is how synchronized actuation instants can be forced in the kernel. For example, a solution could be to use a periodic task for computing u_k and another periodic task for applying u_k at the required time. Another solution could be to enforce synchronized executions at the kernel level, using the EDF tick counter. Moreover, since different solutions to the jitter problem such as [42] or [43] could have also been applied, students more confident or interested in specific fields can select the solution that better meets their preferences.

V. Tentative work plan and its application to a specific course

The previous section has detailed some of the steps required to successfully carry out the lab activity presented in this paper. This section summarizes them in order to propose a tentative work plan that is divided into several sessions, each one being a two-hour lab.

S1 - Introduction: Introduction to the activity, and simulation of the open-loop response after obtaining the state-space form of the $RCRC$ circuit from the circuit differential equations (1) (consider random values for R and C). Here it is assumed that state-space notation is chosen.

S2 - Problem specification (a): This session should be used to specify the problem in terms of the levels for the reference signal and for discrete controller design, which includes selecting the sampling period and closed loop pole locations, if pole placement is used. Other control approaches, like optimal control, could also be used.

S3 - Problem specification (b): To complement the previous session, observers should also be designed and simulated. The outcome of this session should be the complete simulation setup.

S4 - Basic implementation (a): Build the $RCRC$ circuit and verify its dynamics in open-loop. Start the controller implementation in a periodic hard real-time task in the processing platform.

S5 - Basic implementation (b): Finish the controller implementation and test its correctness.

S6 - Multitasking (a): Incorporate a noisy task in the simulation setup to evaluate the effects of jitter. This step would require to use, for example, the TrueTime simulator.

S7 - Multitasking (b): Incorporate the noisy task in the implementation and validate the previous simulation results.

S8 - Advanced implementation (a): If degradation in control performance is detected in the previous session, simulate advanced control algorithms or adopt real-time techniques to solve or reduce the jitter problem.

S9 - Advanced implementation (b): Implement the previous solutions and validate them.

Note that the program timing, layout and the set of covered topics should be adapted to particular needs/background of the target audience or to the goals of the specific curriculum. For example, the proposed activity has been introduced as a part of the curriculum of the 2-year master degree on *Automatic Control and Industrial Electronics* in the Engineering School in Vilanova i la Geltrú (EPSEVG) of the Technical University of Catalonia (UPC) [47]. In particular, since 2007, the experiment was tailored to become part of the laboratory for the Control Engineering course, which covers continuous and discrete linear time invariant (LTI) control systems, as well as non-linear control systems, all using state-space formalism. Sessions S1 to S5 were adopted for the laboratory of the discrete LTI control systems part.

The Control Engineering course can be followed by students either in the first semester of the first year or in the first semester of the second year. Students choosing the second option simultaneously attend a course on real-time systems. Therefore, within the same classroom, not all the students are familiar with real-time systems. To overcome this apparent drawback, teams of three students were formed containing at least a student with competence on real-time systems. Within such heterogeneous teams in terms of skills and theoretical background, it was observed that students took their responsibilities and team-work was significantly improved.

As in every course edition, after finishing the discrete LTI control systems part, a short and simple questionnaire is given to the students to let them evaluate several aspects of this part of the course. The question *Do the laboratory activities permit to better understand the theoretical concepts?* is the only one related to the laboratories. Looking at the students answers, and taking into account that before introducing the presented experiment the lab activities were focused on the simulation of an inverted pendulum, the percentage of students appreciating the practical part has increased significantly. Although the standard course evaluation indicates this positive trend, a more complete evaluation tailored to the introduction of this experiment is required.

VI. Conclusions

This paper has presented a laboratory activity to be integrated in the education curriculum of embedded control systems engineers. The activity consists of a real-time controller of a $RCRC$ electronic circuit. The potential benefits, competences to be acquired, and expected learning outcomes for students have been presented. The selection of the plant and processing platform has been discussed. Extensive details of a sample implementation have been presented and a tentative work plan for carrying out the activity has been provided.

In summary, the proposed activity poses several *real* challenges to the students that can be met by putting together interdisciplinary skills (electronics, real-time systems, control theory, programming) towards a single goal: building a working system.

References

[1] D. J. Jackson and P. Caspi, "Embedded systems education: future directions, initiatives, and cooperation", *SIGBED Rev.*, vol. 2, no. 4, pp. 1-4, Oct. 2005.

[2] A. Crespo, J. Vila, F. Blanes, and I. Ripoll, "Real-time education in a control engineering curriculum," in *Third IEEE Real-Time Systems Education Workshop*, 1998.

[3] K. G. Ricks, D. J. Jackson, and W. A. Stapleton, "Incorporating embedded programming skills into an ECE curriculum", *SIGBED Rev.*, vol. 4, no. 1, pp. 17-26, Jan. 2007.

[4] W. A. Halang,"A curriculum for real-time computer and control systems engineering", *IEEE Transactions on Education*, vol. 33, no. 2, pp. 171-178, May 1990.

[5] P. Caspi, A. San Giovanni-Vincentelli, et al., "Guidelines for a graduate curriculum on embedded software and systems", *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 3, pp. 587 - 611, Aug. 2005.

[6] A. Sangiovanni-Vincentelli and A. Pinto, "An overview of embedded system design education at Berkeley", *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 3, pp. 472-499, Aug. 2005.

[7] K. G. Ricks, D. J. Jackson, and W. A. Stapleton, "An embedded systems curriculum based on the IEEE/ACM model curriculum", *IEEE Transactions on Education*, vol. 51, n. 2, pp. 262-270, May 2008.

[8] D. Davcev, B. Stojkoska, S. Kalajdziski, and K. Trivodaliev, "Project based learning of embedded systems", in *Proceedings of the 2nd WSEAS international Conference on Circuits, Systems, Signal and Telecommunications*, 2008.

[9] S. Nooshabadi and J. Garside, "Modernization of teaching in embedded systems design–An international collaborative project", *IEEE Transactions on Education*, vol. 49, no. 2, pp. 254-262, May 2006.

[10] J. W. Bruce, J. C. Harden, and R. B. Reese, "Cooperative and progressive design experience for embedded systems", *IEEE Transactions on Education*, vol. 47, no. 1, pp. 83-92, Feb. 2004.

[11] J. Fernandez, R. Marin, and R. Wirz, "Online competitions: an open space to improve the learning process," *IEEE Transactions on Industrial Electronics*, vol.54, no.6, pp. 3086-3093, Dec. 2007.

[12] U. Munz, P. Schumm, A. Wiesebrock, and F. Allgower, "Motivation and learning progress through educational games," *IEEE Transactions on Industrial Electronics*, vol.54, no.6, pp. 3141-3144, Dec. 2007.

[13] L. Gomes, and S. Bogosyan, "Current Trends in Remote Laboratories," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 12, pp. 4744-4756, Dec. 2009.

[14] D. T. Rover, R. A. Mercado, Z. Zhang, M. C. Shelley, and D. S. Helvick, "Reflections on teaching and learning in an advanced undergraduate course in embedded systems," *IEEE Transactions on Education*, vol.51, no.3, pp.400-412, Aug. 2008.

[15] K.-E. Årzén, A. Blomdell, and B. Wittenmark, "Laboratories and real-time computing: integrating experiments into control courses," *IEEE Control Systems Magazine*, vol. 25, no. 1, pp. 30-34, Feb. 2005.

[16] G. Buttazzo, "Research trends in real-time computing for embedded systems", *ACM SIGBED Review*, vol. 3, no. 3, Jul. 2006.

[17] P. Horacek, "Laboratory experiments for control theory courses: A survey," *Annual Reviews in Control*, vol. 24, pp. 151162, 2000.

[18] M. Moallem, "A laboratory testbed for embedded computer control," *IEEE Transactions on Education*, vol. 47, no. 3, pp. 340-347, Aug. 2004.

[19] D.-J. Lim, "A laboratory course in real-time software for the control of dynamic systems", *IEEE Transactions on Education*, vol. 49, no. 3, pp. 346-354, Aug. 2006.

[20] M. Huba and M. Simunek, "Modular approach to teaching PID control," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 6, pp. 3112-3121, Dec. 2007.

[21] VxWorks operating system from WindRiver, http://www.windriver.com/

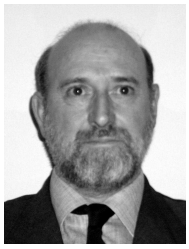
[22] R. Marau, P. Leite, M. Velasco, P. Martí, L. Almeida, P. Pedreiras, and J. M. Fuertes, "Performing flexible control on low cost microcontrollers using a minimal real-time kernel", *IEEE Transactions on Industrial Informatics*, vol. 4, no. 2, pp. 125-133, May 2008.

[23] F. Salewski, D. Wilking, and S. Kowalewski, "Diverse hardware platforms in embedded systems lab courses: a way to teach the differences", *SIGBED Rev.*, vol. 2, no. 4, pp. 70-74, Oct. 2005.

[24] Evidence srl., http://www.evidence.eu.com/
[25] Real-time Linux Foundation, Inc., http://www.realtimelinuxfoundation.org/
[26] J. A. Stankovic and K. Ramamritham, "The spring kernel: a new paradigm for real-time operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 23, no. 3, pp. 54–71, 1989.
[27] K. M. Zuberi, P. Pillai, and K. G. Shin, "Emeralds: a small-memory real-time microkernel," in *7th ACM symposium on Operating Systems Principles*, pp. 277–299, 1999.
[28] P. Gai, G. Lipari, L. Abeni, M. di Natale, and E. Bini, "Architecture for a portable open source real-time kernel environment," in *Proceedings of the Second Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial*, Nov. 2000.
[29] E. Mumolo, M. Nolich, and M. Noser, "A hard real-time kernel for motorola microcontrollers," in *23rd International Conference on Information Technology Interfaces*, Jun. 2001.
[30] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo, "A new kernel approach for modular real-time systems development," in *13th IEEE Euromicro Conference on Real-Time Systems*, Jun. 2001.
[31] D. Henriksson and A. Cervin, "Multirate feedback control using the TinyRealTime kernel," in *Proceedings of the 19th International Symposium on Computer and Information Sciences*, Antalya, Turkey, Oct. 2004.
[32] OSEK/VDX Portal, http://www.osek-vdx.org/.
[33] G. Buttanzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Norwell, MA, USA: Kluwer Academic Publishers, 1997.
[34] Eclipse portal, http://www.eclipse.org/.
[35] Microchip portal, http://www.microchip.com/.
[36] Scilab/Scicos portal, http://www.scilab.org/.
[37] MathWorks portal, http://www.mathworks.com/.
[38] D. Hercog, B. Gergic, S. Uran, and K. Jezernik, "A DSP-based remote control laboratory," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 6, pp. 3057-3068, Dec. 2007.
[39] K. Ogata, *Modern Control Engineering*, 4th Edition, Prentice Hall, 2001.
[40] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[41] K.-E. Årzén, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *39th IEEE Conference on Decision and Control*, 2000.
[42] J. Skaf and S. Boyd, "Analysis and synthesis of state-feedback controllers with timing jitter," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 652-657, March 2009.
[43] P. Balbastre, I. Ripoll, J. Vidal, and A. Crespo, "A task model to reduce control delays," *Real-Time Systems*, vol. 27, no. 3, pp. 215-236, Sep. 2004.
[44] C. Lozoya, M. Velasco, and P. Martí, "The one-shot task model for robust real-time embedded control systems", *IEEE Transactions on Industrial Informatics*, vol. 4, no. 3, pp. 164-174, Aug. 2008.
[45] D. Henriksson, A. Cervin, and K.-E. Årzén, "TrueTime: simulation of control loops under shared computer resources," in *15th IFAC World Congress*, 2002.
[46] Y. Wu, E. Bini, and G. Buttazzo, "A framework for designing embedded real-time controllers", in *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug. 2008.
[47] EPSEVG School portal, http://www.epsevg.upc.edu/.

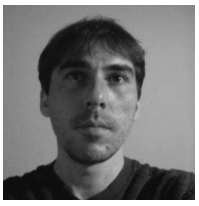
Pau Martí (M'02) received the degree in computer science and the Ph.D. degree in automatic control from the Technical University of Catalonia, Barcelona, Spain, in 1996 and 2002, respectively. Since 1996, he has been an assistant professor in the Department of Automatic Control at the Technical University of Catalonia. During his Ph.D. he was a visiting student at Mälardalen University, Västerås, Sweden. From 2003 to 2004, he held a research fellow appointment in the Computer Science Department at the University of California at Santa Cruz. His research interests include embedded systems, control systems, real-time systems, and communication systems.

Manel Velasco graduated in maritime engineering in 1999 and received the Ph.D. degree in automatic control in 2006, both from the Technical University of Catalonia, Barcelona, Spain. Since 2002, he has been an assistant professor in the Department of Automatic Control at the Technical University of Catalonia. His research interests include artificial intelligence, real-time control systems, and collaborative control systems, especially on redundant controllers and multiple controllers with self-interacting systems.



Josep M. Fuertes (S'74-M'96) received the degree in industrial engineering and the Ph.D. degree from the Technical University of Catalonia (UPC), Barcelona, in 1976 and 1986, respectively.

From 1975 to 1986, he was a Researcher at the Institut de Cibernética (Spanish Consejo Superior de Investigaciones Cientficas). In 1987, he became a Permanent Professor with the UPC. In 1987, he got a position for a year at the Lawrence Berkeley Laboratory, Berkeley, CA, where he was a Visiting Scientific Fellow for the design of the Active Control System of the W. M. Keck 10-m segmented telescope (Hawaii). From 1992, he was the responsible of the University Research Line in Advanced Control Systems and later of the Distributed Control Systems Group. From 1996 to 2001, he acted as a Spanish Representative at the Council of the European Union Control Association. His research interests are in the areas of distributed, networked and real-time control systems and applications. Since 1989, he has been coordinating projects at the national and international levels related to the aforementioned areas of expertise. He is a member of the Administrative Committee of the IEEE Industrial Electronics Society, where he is chairing the Technical Committee of Networked Based Control Systems and Applications. He has collaborated as the Organizer, Chairman, Session Organizer, and member of program committees for several international conferences.



Antonio Camacho received the B.S. degree in chemical engineering in 2000 and the M.S. degree in automation and industrial electronics in 2009, both from the Technical University of Catalonia, Barcelona, Spain. Currently, he is pursuing the Ph.D. degree in electronic engineering also at the Technical University of Catalonia. His research interests include networked and embedded control systems, industrial informatics, and power electronics.

Giorgio Buttazzo (SM'05) received the degree in electronic engineering from the University of Pisa, Pisa, Italy, in 1985, the master degree in computer science from the University of Pennsylvania, Philadelphia, in 1987, and the Ph.D. degree in computer engineering from the Scuola Superiore SantAnna of Pisa in 1991.

He is a Full Professor of computer engineering at the Scuola Superiore SantAnna of Pisa. From 1987 to 1988, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He has authored six books on real-time systems and over 200 papers in the field of real-time systems, robotics, and neural networks.