Designing a Practical Course in Networked Control Systems

Josep M. Fuertes, Ricard Villà, Jordi Ayza, Pere Marés, Pau Martí, Manel Velasco, José Yépez, Gina Torres, Miquel Perelló

***Abstract*—This paper presents a hands-on course in networked control systems (NCS) to be integrated in the education of embedded control systems engineers. The course activities have a strong practical component and most of them are applied exercises to be implemented in a NCS setup. The paper describes the experimental setup and then proposes several activities that can be shaped into a course program according to the needs and diverse background of the targeted audience.**

***Index Terms*—Embedded systems education, networked control systems.**

I. INTRODUCTION

Networked control systems [1], [2], i.e. control loops closed over communication networks where sensors, controllers and actuators are physically distributed and exchange control data through a shared network, are gaining increased attention in many control application areas due to their cost-effectiveness, reduced weight and power requirements, simple installation and maintenance, and high reliability. At the same time, the underlying required control theory is starting to offer mature and methodological results, e.g. [3], [4].

In a parallel track, since the economic importance of embedded systems has grown exponentially as electronic components are in everyday use devices, embedded systems education is becoming an strategic asset. Hence, university curricula are being adapted accordingly to cover this domain [5]. In addition, noting that many embedded systems are control systems [6] and considering the importance of NCS in industrial processes, there is a growing demand of including NCS courses in the education of embedded systems engineers.

The traditional teaching approach to the diverse disciplines involved in NCS such as control systems, real-time computing and communication systems, can be often quite math-intensive and abstract, thus failing to introduce students to the realities of NCS implementation. Hence, laboratory activities are crucial to consolidate the diverse theoretical material. Following this trend, this paper presents a hands-on course in networked control systems to be integrated in the education of embedded control systems engineers. The course activities have a strong practical component and most of them are applied exercises to be implemented in a NCS setup. This course can be taken as a complimentary material to other exiting courses in NCS and to other initiatives related to NCS education such as the ”Networked Control Systems Repository” at http://filer.case.edu/org/ncs/index.htm, the NCS wiki page course at http://www.cds.caltech.edu/~murray/wiki/index.php/EECI08:_Introduction_to_Networked_Control_Systems, or other similar efforts. The proposal in this paper extends to a networked setup a previously presented laboratory experiment targeting microprocessor-based real-time control systems [7].

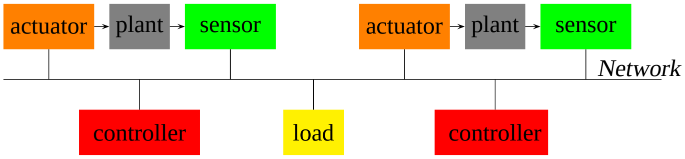


Fig. 1. Networked control system scheme

After describing the basics on NCS (Section II), the paper describes the experimental setup from a hardware (Section III) and software point of view (Section IV), and then proposes several activities (Section V) that can be shaped into a course program according to the needs and diverse background of the targeted audience. Section VI concludes the paper.

II. BACKGROUND ON NCS

NCS take different forms, and two major types of control systems can be identified: shared-network control systems and remote control systems. The hands-on course proposal targets the first type, although several concepts can also be applied to the second type.

Hence, the course context is the NCS illustrated in Figure 1. Several control loops, each one formed by a sensor, a controller and an actuator implemented in physically separated nodes, share a single broadcast domain to exchange the control data required for each control loop operation. In addition, other nodes, represented by the load boxes, also use the network to exchange other non-control data.

For a given networked closed loop system, a control job will denote the required operations for each plant update. Hence, each control job would basically require sending the sensor reading in the sensor message after sampling the plant, and sending the control signal in the control message after computing its value in the controller node using the information contained in the incoming sensor message. Thus, in terms of bandwidth utilization, each control job simplifies to sending two messages, the sensor and the control message.

This work was partially supported by Spanish Ministerio de Educación y Ciencia Project ref. CICYT DPI2010-18601.

The authors are with the Automatic Control Department, Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, Spain. E-mail: josep.m.fuertes@upc.edu

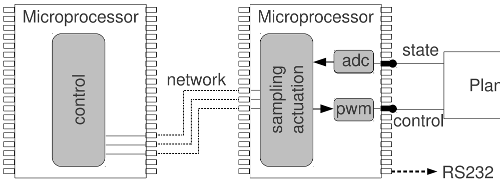


Fig. 2. Experimental setup.

Component	Value
R_3	$1k\Omega$
R_1	$100k\Omega$
R_2	$100k\Omega$
C_1	$470nF$
C_2	$470nF$

TABLE I
ELECTRONIC COMPONENTS VALUES

The most common design and implementation approach for NCS consists in the periodic execution of the control algorithm, which implies that messages from sensors to controllers and from controllers to actuators are periodic [8]. The course will cover these methods, but will also takle other approaches that go beyond the periodicity of the standard approach. Among them, two new tendencies for the analysis and design of NCS can be identified. The first one is to apply rate adaptation techniques where the period is selected according to the controlled system dynamics and/or to the bandwidth conditions, e.g. [9]–[11]. The main goal of these approaches is to improve the aggregated control performance for the set of control loops by efficiently using all the communication bandwidth. The second tendency is to apply event-based sampling techniques which produce non-periodic executions of the control algorithm, and therefore, non-periodic messaging in the network, e.g. [12]–[14]. The main goal of these approaches is to minimize the bandwidth utilization while still guaranteeing stability and acceptable control performance.

III. EXPERIMENTAL SET-UP

A. Introduction

The desired scheme for each experimental setup is illustrated in Figure 4. Hence, each student (or student group) will have a two-node NCS in which one node acts as a controller (left node) while the other node acts as a sensor and actuator (right node) and is attached to the plant. The reason for putting together the sensor and actuator in the same node is to save hardware resources.

The controlled plant and processing platform (hardware, real-time operating system, and network) have been carefully selected to have a friendly, flexible, and powerful experimental set-up.

B. 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.

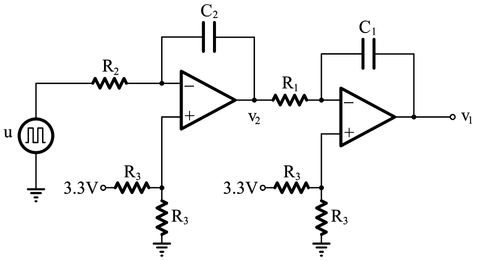


Fig. 3. Plant: electronic double integrator (DI) circuit

Following the same reasons discussed in [7], a simple electronic circuit in the form of a double integrator (Figure 3) was selected[1]. The relative simplicity of its components together with the inherent unstable dynamics that makes the control more challenging have been the main reasons for its selection. Note however that experiments using other plants can be complementary to the approach presented here.

The selection of an electronic circuit as a plant has also an important advantage: depending on the specific circuit, it can be directly plugged into a micro-controller without using intermediate electronic component. 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 modeling effort to study the plant and no models for actuators or sensors are required.

The DI nominal electronic components are shown in table I.

The operational amplifier in integration configuration [15] can be model by

$$V_{\text{out}} = \int_0^t -\frac{V_{\text{in}}}{RC}\,dt + V_{\text{initial}} \tag{1}$$

where V_{initial} is the output voltage of the integrator at time $t = 0$, and ideally $V_{\text{initial}} = 0$, and V_{in} and V_{out} are the input and output voltages of the integrator, respectively.

Taking into account (1), and the scheme shown in Figure 3, the double integrator plant dynamics can be modeled by

$$\begin{aligned} \frac{dv_2}{dt} &= \frac{-1}{R_2C_2}u \\ \frac{dv_1}{dt} &= \frac{-1}{R_1C_1}v_2 \end{aligned}$$

---

[1] Note that in the integrator configuration, the operational amplifiers require positive and negative input voltages. Otherwise, they will quickly saturate. However, since the circuit is powered by the micro-controller, and thus no negative voltages are available, the 0V voltage (V_{ss}) in the non-inverting input has been shifted from GND to half of the value of V_{cc} (3.3V) by using a voltage divider R_3. Therefore, the operational amplifier differential input voltage can take positives or negatives values.

Component	Value
R_1	$100k\Omega$
R_2	$100k\Omega$
C_1	$420nF$
C_2	$420nF$

TABLE II
VALIDATED VALUES FOR THE ELECTRONICS COMPONENTS.

In state space form, the model is

$$\begin{aligned} \begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} &= \begin{bmatrix} 0 & \frac{-1}{R_1 C_1} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-1}{R_2 C_2} \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \end{aligned}$$

Taking into account the tolerances in the electronics components (5% for resistors and 25% for capacitors), the model that best adapts to the real plant is given by the values listed in table II. Hence, with the validated values for the components, the model is given by

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & -23.809524 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -23.809524 \end{bmatrix} u \quad (2) \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x \end{aligned}$$

Note that the plant is unstable because the eigenvalues of the system matrix are $\lambda_{1,2} = 0$. The goal of the controller is to make the circuit output voltage (v_1 in Figure 3) to track a reference signal by giving the appropriate voltage levels (control signals) u. Both states v_1 and v_2 can be read via the Analog-to-Digital-Converter (ADC) port of the micro-controller and u is applied to the plant through the Pulse-Width-Modulation (PWM) port.

C. Processing platform

The processing platform consists of the hardware platform, the real-time operating system and the network. As hardware platform, a micro-controller based architecture was selected because NCS are typically implemented using this type of hardware. As discussed in [7], for the processing platform adopting the Flex board [16] (in its full version) equipped with a Microchip dsPIC DSC micro-controller dsPIC33FJ256MC710 represents a good compromise between cost, processing power, and programming flexibility. And regarding the real-time operating system, Erika Enterprise real-time kernel [16] was selected because it provides full support to the Flex board in terms of drivers, libraries, programming facilities, and sample applications, and it gives support for preemptive and non-preemptive multitasking, and implements several scheduling algorithms [17]. In addition, its API provides support for tasks, events, alarms, resources, application modes, semaphores, and error handling, permitting to enforce real-time constraints to application tasks to show students the effects of sampling periods, delays and jitter on control performance.

Regarding the network, the CAN (Controller Area Network, [18]) protocol that was originally designed for the automotive industry, but it has also become a popular bus in industrial automation as well as other applications has been selected. CAN provides the basis for many cost-effective distributed embedded applications, and its properties and functionality such as reliability, dependability, or clock synchronization, are being constantly enhanced.

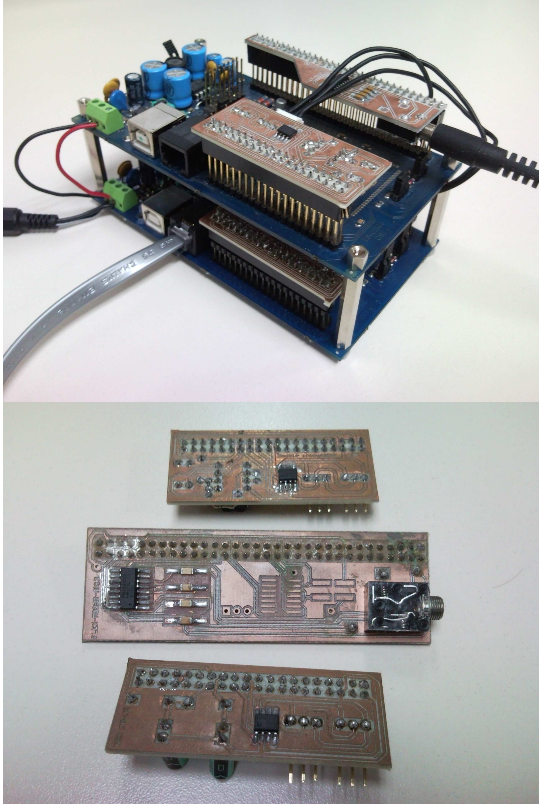


Fig. 4. Controller-sensor/actuator hardware (top). Details of the CAN, DI and RS232 daughter boards (bottom)

Once the platform is ready, the networked setup for each student looks like as in Figure 4 (top). The bottom board acts as a (remote) controller, and communicates via CAN with the top board, that acts as a sampler and actuator. Note that the same hardware, micro-processor based control can also be tested. In this case, the bottom board is not used, and the top board performs all the activities: sampling, control algorithm computation and actuation. The daughter boards plugged into the top and bottom board, illustrated in Figure 4 (bottom), serve different objectives: the daughter board in the bottom is for CAN communication, the one in the middle is for RS232 communication (for monitoring) and the top daughter board is the DI circuit also enabled with CAN communication.

It is interesting to note that the modular architecture design permits to network all the different students setups in a single CAN network. In this way, a full networked control system can be built, where several nodes acting as controllers, sensors and actuators control different double integrator systems. This richer setup will permit to observe the interaction among different closed-loops in terms of bandwidth utilization and control performance.

In addition to all the pair of controller and sensor/actuator nodes that constitute several loops closed over the network, another node can be added to the network acting as a mon-

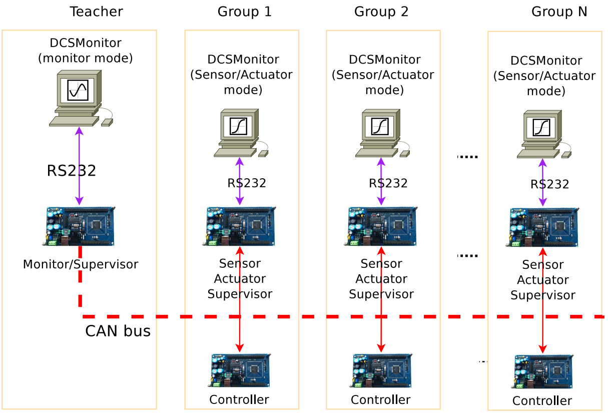

Teacher
Group 1
Group 2
Group N
DCSMonitor (monitor mode)
RS232
Monitor/Supervisor
CAN bus
DCSMonitor (Sensor/Actuator mode)
RS232
Sensor Actuator Supervisor
Controller

Fig. 5. Scheme of the overall networked control system.

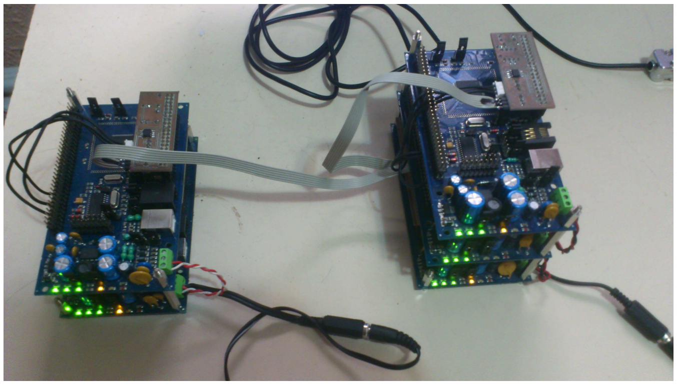


Fig. 6. Example of networked control system.

```
CPU mySystem {
  OS myOs {
    EE_OPT = "DEBUG"; CPU_DATA = PIC30 {
      APP_SRC = "setup.c"; APP_SRC = "e_can1.c";
      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 TaskReferenceChange {
      REL_DEADLINE = "0.005s"; PRIORITY = 3;
      STACK = SHARED;SCHEDULE = FULL;};
    TASK TaskController {
      REL_DEADLINE = "0.05s"; PRIORITY = 3;
      STACK = SHARED; SCHEDULE = FULL;};
    COUNTER myCounter;
    ALARM AlarmReferenceChange {
      COUNTER = "myCounter";
      ACTION = ACTIVATETASK {
        TASK = "TaskReferenceChange"; };};
    ALARM AlarmController {
      COUNTER = "myCounter";
      ACTION = ACTIVATETASK {
        TASK = "TaskController"; };};
};
```

Fig. 7. Configuration file (conf.oil) for the controller node

itor/supervisor. The hardware of this node is again the full Flex board, equipped with the daughter board with CAN communication and the RS232 board used for debugging purposes. The role of this node is to gather information of the state of all the control loops, as well as to monitor and manage the network bandwidth. This node would be mainly used for the instructor/teacher, although it can be also used by students.

The complete scheme showing N control loops together with the supervisor/monitor is illustrated in Figure 5. In the figure, each group is a pair of controller and sensor/actuator nodes that are connected using CAN (that would correspond to the setup shown in Figure 4 top) and it is the hardware setup for each student (or group of students). Then, all the groups can be networked between them, using also CAN, and the monitor/supervisor node can also be attached to the network.

Figure 6 shows an example of the full NCS with two groups, together with the monitor. In this case, one of the groups has the controller-sensor/actuator hardware and the monitor hardware plugged together in a single tower (on the right).

IV. SOFTWARE

This section briefly describes the main software components that have been prepared for the experimental setup. Two types can be distinguished. First, the software that goes to each node (Flex board) and second, the software that can be run in an external PC for debugging purposes, and that is called "DCSMonitor" in Figure 5. The node codes, the DCSMonitor, and other information related to this hands-on course are available at http://code.google.com/p/pfc-platform-test/source/browse/.

A. *Main software in NCS nodes*

The description of the Erika codes is ordered according to the three type of nodes: controller, sensor/actuator, monitor/supervisor. For each node, the kernel configuration file *conf.oil* specifies the main parameters for the dsPIC and real-time kernel, and the *code.c* contains the main functionality.

1) Controller node: The configuration file is shown in Figure 7. It specifies the C files that are used in this node, the scheduling algorithm (EDF, Earliest Deadline First), and it defines two tasks, `TaskReferenceChange` and `TaskController`, which are implemented in the *code.c* and associated to an alarm to control their periodicity of execution.

The *code.c* file in the node has three main parts (Figure 8): the `main`, and the two task defined in the *conf.oil*. The `main` configures the `TaskReferenceChange` whose role is to create the reference signal to be tracked by the controller. The `TaskReferenceChange` code simply generates a square wave that switches from -0.5v to -0.5v each second, and this information is sent over CAN for debugging purposes. The `TaskController` is activated by interrupt upon reception of the sensor message delivered over CAN. It gets the plant state for the message, implements a control law (in the figure a state-feedback controller with a tracking configuration), and sends the computed control signal over CAN in the control message.

2) Sensor/Actuator node: The sensor/actuator node configuration file is very similar to the one in the controller

```
TASK(TaskReferenceChange){
  if (r == -0.5){r=0.5; LATBbits.LATB14 = 1;
  }else{ r=-0.5; LATBbits.LATB14 = 0;}
  Send_Controller_ref_message(&r);
}
TASK(TaskController){
  x0=*(p_x0);//Get state x[0] from CAN msg
  x1=*(p_x1);//Get state x[1] from CAN msg
  x_hat[0]=x0-r*Nx[0]; x_hat[1]=x1-r*Nx[1];
  u_ss=r*Nu;
  u=-k[0]*x_hat[0]-k[1]*x_hat[1]+u_ss;
  Send_Controller2Actuator_message(&u);
}
int main(void){
  Sys_init();
  SetRelAlarm(AlarmReferenceChange,1000,1000);
  for (;;); return 0;
}
```

Fig. 8. Main parts of the code.c for the controller node

```
TASK(TaskSensor){
  LATBbits.LATB14 ^= 1; Read_State();
  Send_Sensor2Controller_message(&x[0]);
}
TASK(TaskSensor_supervision){
  LATBbits.LATB14 ^= 1; Read_State();
  Send_Sensor2Supervisor_message(&x[0]);
}
TASK(TaskActuator){
  u=(*p_u); Get state u from CAN msg
  PDC1=((*(p_u))/v_max)*0x7fff+0x3FFF;
}
TASK(TaskSupervision){
/* It sends data via RS-232 to the PC
}
int main(void){
  Sys_init();
  SetRelAlarm(AlarmSensor,1000,50);
  SetRelAlarm(AlarmSupervision, 1000, 10);
  SetRelAlarm(AlarmSensor_supervision, 1000, 10);
  for (;;); return 0;
}
```

Fig. 9. Main parts of the code.c for the sensor/actuator node

```
TASK(TaskControllerMonitor){
  x0=*(p_x0);//Get state x[0] from CAN msg
  x1=*(p_x1);//Get state x[1] from CAN msg
}
TASK(TaskSensor_supervision){
  x0=*(p2_x0);//Get state x[0] from CAN msg
  x1=*(p2_x1);//Get state x[1] from CAN msg
}
TASK(TaskActuatorMonitor){
  u=(*p_u);
  x0=*(p_x0);//Get state x[0] from CAN msg
  x1=*(p_x1);//Get state x[1] from CAN msg
}
TASK(TaskSupervision){
/* It sends data via RS-232 to the PC
}
TASK(TaskToggleLed){
  LATBbits.LATB14 ^= 1;//Toggle orange led
}
TASK(TaskCANUseless){
  Send_CAN_useless();
}
int main(void){
Sys_init(); init_devices_list();
for (;;); return 0;
}
```

Fig. 10. Main parts of the code.c for the monitor/supervisor node

node. However, it defines different tasks: `TaskSensor`, `TaskSensor_supervision`, `TaskActuator`, `TaskSupervision`, that are code in the corresponding *code.c* file.

Figure 9 shows the main parts of the *code.c* file. In the `main`, the periodicity for the sensor task is defined to 50ms and the periodicity for the supervision tasks are defined to 10ms. Note that the periodicity of the actuator task is not defined because it is executed upon reception of the control message send by the controller node. The `TaskSensor` reads the plant state and sends this value in the sensor message over CAN. The `TaskSensor_supervision` sends the plant state over CAN, that will be used for debugging purposes. Similarly, the `TaskSupervision` sends a similar information through the RS232 port, also for debugging purposes. Finally, the TaskActuator, upon reception of the control message, takes the control signal form the message, and applies it to the plant using the PWM.

3) Monitor/supervision: This node is the special purpose node that gathers information from all the control loops and network. It communicates with the DCSMonitor software that runs in an external PC, that will be described in next section.

The *conf.oil* for this node defines the `TaskSupervision`, `TaskActuatorMonitor`, `TaskControllerMonitor`, `TaskSensor_supervision`, `TaskToggleLed` and the `TaskCANUseless`, which are coded in the *code.c* file.

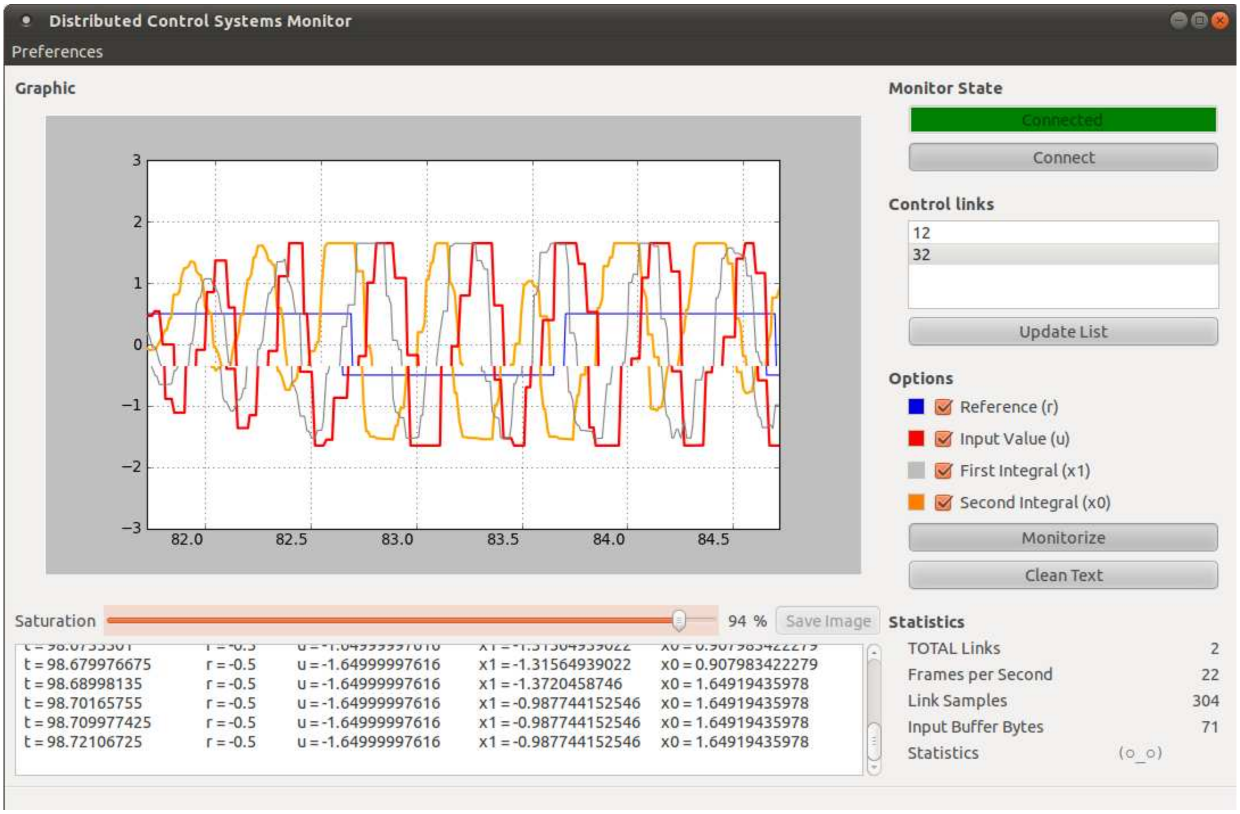
The `TaskSupervision` sends data over the RS232 port. The `TaskActuatorMonitor` mainly obtains the control signal for a given control loop, while the tasks `TaskControllerMonitor`, `TaskSensor_supervision` obtain the state of a given plant. The `TaskToggleLed` blinks a led, and the `TaskCANUseless` is used to regulate the bandwidth of the network by sending dummy messages.

Although not detailed here, the full *code.c* also implements the code that manages the interaction between the monitor/supervision node and the DCSMontior software, which communicate over RS232.

B. DCSMonitor

The DCSMonitor is a monitoring program that can be used by each student group to monitor its control loop dynamics, but it has been mainly designed for the instructor/teacher to allow monitoring any of the group control loops, as well as to regulate the network bandwidth.

Figure 11 provides a general view of the DCSMonitor. It can perform three main activities. First, it permits to monitor the number of control loops that are exchanging data over the network (Control links in the figure). It also displays



```
t = 98.679976675     r = -0.5     u = -1.64999997616     x1 = -1.31564939022     x0 = 0.907983422279
t = 98.68998135      r = -0.5     u = -1.64999997616     x1 = -1.3720458746      x0 = 1.64919435978
t = 98.70165755      r = -0.5     u = -1.64999997616     x1 = -0.987744152546    x0 = 1.64919435978
t = 98.709977425     r = -0.5     u = -1.64999997616     x1 = -0.987744152546    x0 = 1.64919435978
t = 98.72106725      r = -0.5     u = -1.64999997616     x1 = -0.987744152546    x0 = 1.64919435978
```

TOTAL Links	2
Frames per Second	22
Link Samples	304
Input Buffer Bytes	71
Statistics	(o_o)

Fig. 11. DCSMonitor.

Signal	Value	Meaning
SIGNAL_STOP	0x01	It cancels current actions
SIGNAL_MONITOR	0x00	To monitor a particular control loop
SIGNAL_PERCENT	0x02	To generate artificial load in the bus
SIGNAL_DEVICES	0x04	To list all active control loops

TABLE III
Commands from DCSMonitor to the monitor/supervisor node

the dynamics (reference signal, control signal and states) of a given control loop, either numerically or graphically. And finally, it has a slider bar that permits to regulate the number of dummy messages that are sent over the network to create different bandwidth loads.

To allow this functionality, the communication between the monitor/supervisor node and the DCSMonitor software over RS232 was configured two be bidirectional. On one hand, the DCSMonitor sends control commands to the monitor/supervisor node using a simple 8-byte frame coded according to an identifier, whose values and meaning are summarized in table III. On the other hand, the monitor/supervisor node sends to the DCSMonitor a more complete 71-byte frame in which the monitor/supervisor node communicates information about the control loop under supervision, as well as the list of active control loops if required. In the case that the DCSMonitor is used for a student group to monitor its own control loop dynamics, the information is sent by the sensor/actuator node of that particular control loop using a medium 23-byte frame. A scheme of this data exchange is shown in Figure 12, where the top sub-figure shows the communication when the DCSMonitor is used by the teacher while the bottom sub-figure shows the communication when the DCSMonitor is used by a group of students.

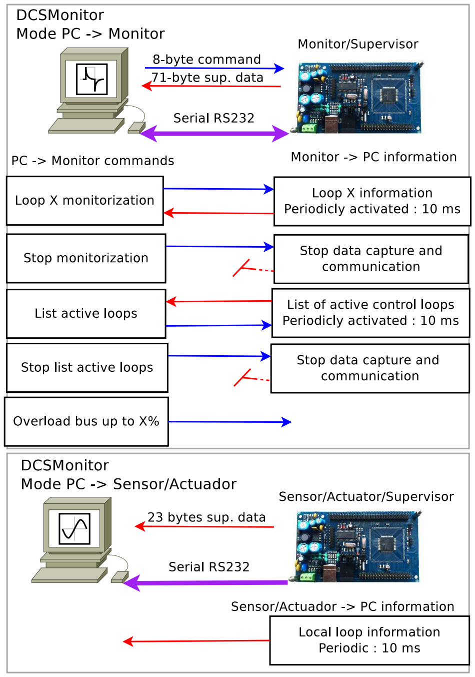


Fig. 12. RS232 communication between the monitor/supervisor node and the DCSMonitor. Top: teacher setup. Bottom: student group setup.

In order to allow the monitor/supervisor node to send

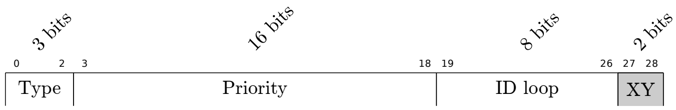


Fig. 13. CAN message codification.

accurate monitoring information to the DCSMonitor software, the CAN messages used in each control loop were coded following different guidelines. First, different levels of priority were required. In addition, it was required to be able to identify different control loops, and in each control loop, different message classes, and even subclasses. This has been achieved by coding each message identifier as shown in Figure 13.

The message type field permits to distinguish a) control messages (000) that are sent from any controller node to its sensor/actuator node and must be of higher priority, b) sensor messages (001) that are sent from any sensor/actuator node to its controller node, and c) general purpose messages (010) that depending on the subclass specifies reference/change messages (00) that are sent by the controller, and supervisory messages (01) that are sent by the sensor. The latter class is non-critical communication, and therefore, they have the lowest priority. Using this coding, and filling up each data frame with the appropriate information, the monitor/supervisor node can gather all the plants information to be send to the DCSMonitor whenever required.

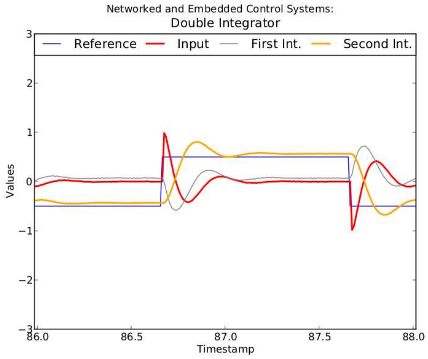
V. COURSE ACTIVITIES

This section presents some of the activities to carry out in the hands on course using the presented setup. To start with, basic control theory may be needed to establish a common knowledge among all the students. It could include

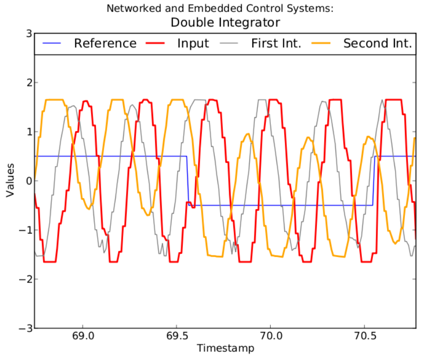
- State-space system representation, in the continuous-time but more important in the discrete-time domain, and including delays
- Discrete-time systems analysis, including topics such as controllability and observability, as well as sampling period selection
- Control design by state feedback, stating with pole placement techniques and considering delays, together with observer and tracking structures

From this basic theory, simulation and implementation exercises including not-NCS and NCS could be for example

1) **Open loop analysis (not-NCS):** Using for example Matlab/Simulink, to obtain the open loop system response of the double integrator circuit in front of a reference signal in the form of a square wave of amplitude from 1.5V to 2V and frequency 1Hz. And to perform a simple stability analysis by pole inspection. This can also be implemented in the hardware set-up, using only the sensor/actuator node acting also as a controller.
2) **Closed-loop design in the continuous-time domain (not-NCS):** To perform state feedback controller design via pole location in the continuous-time domain with tracking structure and assuming that both states can be measured, such that a stable circuit response is achieved and the control signal values range withing 3.38V and 0V. The first constraint is a control specification while the second one is a hardware limitation.
3) **Closed-loop design in the discrete-time domain (not-NCS):** To carry out the state feedback controller design via pole location in the discrete-time domain with tracking structure and assuming that both states x_1 and x_2 are measured, such that the stable circuit response is achieved while the previous hardware constraint is still met. In this case, the sampling period and closed-loop pole locations for controller are can be obtained by standard rules-of-thumb. Alternatively, it can be specified a period of $h = 0.05$ s, and to design a discrete state feedback controller placing the continuous closed loop poles at $p_{1,2} = -5 \pm 20i$, which meet the specifications. Again, this can also be implemented in the hardware set-up, using only the sensor/actuator node acting also as a controller. At this stage, different type of observers can be also designed, simulated and implemented.
4) **Delay effects in a single NCS (NCS)**: To simulate for example with TrueTime (http://www3.control.lth.se/truetime/) the NCS for the double integrator system using the settings for period and desired poles as before, with sampling occurring in the sampler node, control algorithm executing in a controller node, and actuation taking place in the actuator node. As a network, use for example CAN, with different baud-rates and then to analyze, by observing different responses, how the communication delay affects the performance considering that the controller that applies is the one obtained previously. Note that in this case, the delay is constant. The simulation study can also be implemented by each group, locally, using the NCS setup, and monitoring the response in the DCSMonitor.
5) **Delay effects in a single NCS with different traffic loads (NCS)**: To extend the previous simulation, adding more nodes sending additional traffic to the network and observe the effect of this new traffic on the performance of the double integrator control. Note that in this case, the delay varies. This simulation can be also implemented by each group, and in this case, the slider bar fixing the network load in the DCSMonitor can be used to create different network conditions.
6) **NCS simple controller design (NCS)**: The degrading effects of delays can be treated using different approaches. A simple approach is to design the controller assuming an input delay in the system, which will result in extending the original state space system with a new state variable that represents the previous control signal [19]. This imposes to place a third discrete-time pole for obtaining the state-feedback gain. It can be placed at 0. Then it can be noted that the application of this new controller is effective for a constant delay but not for the scenario of varying delay. The simulation can also be implemented in the setup. In addition, all the loops from all the students can be networked together, to see with a more realistic setting the effects of different traffic on control performance. The CAN bus baud-rate



(a) when bandwidth utilization is low



(b) when bandwidth utilization is high

Fig. 14. DI dynamics.

can also be used for experimental purposes.

7) **NCS advanced controller design (NCS)**: In order to design a controller capable of dealing with varying time delays, several strategies are available, including control-centered approaches, see e.g. [1], or implementation-based approaches, such as [20]. One or more of them can be simulated and implemented.
8) **Rate adaptation techniques and event-driven NCS (NCS)**: using the latest results on these areas (see section II), existing results can also be simulated and implemented.

Note that the last exercises can have different levels of difficulty and may require non-basic control and real-time systems theory for its correct simulation and implementation.

Performing the previous activities students can implement a rich set of exercises. They will observe several DI dynamics according to the activity. Just for illustrative purposes, Figure 14 shows the DI response in the case of having different loads in the CAN network.

VI. CONCLUSIONS

This paper has presented a hands-on course to be integrated in the education curriculum of embedded control systems engineers. The main focus of the course is NCS. The selection of the plant and processing platform has been discussed. Details of the code to be executed in each networked node have been presented. And a new software to be executed in an external PC for monitoring purposes has been explained. Finally, a set of course activities have been listed.

In summary, the proposed course 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 networked control system.

REFERENCES

[1] J.P. Hespanha, P. Naghshtabrizi and Yonggang Xu, “A Survey of Recent Results in Networked Control Systems,” *Proceedings of the IEEE* , vol.95, no.1, pp.138-162, Jan. 2007

[2] R.A. Gupta and M.-Y. Chow , “Networked Control System: Overview and Research Trends,” *IEEE Transactions on Industrial Electronics* , vol.57, no.7, pp.2527-2535, July 2010

[3] D. Nešić and D. Liberzon, “A unified framework for design and analysis of networked and quantized control systems,” *IEEE Transactions on Automatic Control*, Vol. 54, No. 4, pp. 732–747, April 2009.

[4] W.P.M.H. Heemels, A.R. Teel, N. van de Wouw, D. Nešić , “Networked Control Systems With Communication Constraints: Tradeoffs Between Transmission Intervals, Delays and Performance,” *IEEE Transactions on Automatic Control*, vol.55, no.8, pp.1781-1796, Aug. 2010

[5] 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.

[6] 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.

[7] P. Martí, M. Velasco,J.M. Fuertes, A. Camacho, G. Buttazzo, “Design of an Embedded Control System Laboratory Experiment,” *IEEE Transactions on Industrial Electronics*, vol.57, no.10, pp.3297-3307, Oct. 2010

[8] D. Hristu-Varsakelis and W. S. Levine, *Handbook of Networked and Embedded Control Systems*, Birkhäuser Boston, June, 2008.

[9] Rehbinder, H. and Sanfridson, M., “Scheduling of a limited communication channel for optimal control,” *Automatica*, vol. 40, n. 3, pp. 491-500, March 2004.

[10] Ben Gaid, M.E.M.; Cela, A.; Hamam, Y., “Optimal integrated control and scheduling of networked control systems with communication constraints: application to a car suspension system,” *IEEE Transactions on Control Systems Technology*, vol.14, no.4, pp. 776-787, July 2006.

[11] P. Martí, A. Camacho, M. Velasco, M. El Mongi Ben Gaid, “Runtime Allocation of Optional Control Jobs to a Set of CAN-Based Networked Control Systems,” *IEEE Transactions on Industrial Informatics*, vol.6, no.4, pp.503-520, Nov. 2010

[12] Hristu-Varsakelis, D., and Kumar, P.R., “Interrupt-based feedback control over a shared communication medium,” *41st IEEE Conference on Decision and Control*, Dec. 2002.

[13] X. Wang and M. Lemmon, “Decentralized Event-triggering Broadcast over Networked Systems,” *Hybrid Systems: Computation and Control*, 2008.

[14] A. Anta and P. Tabuada, “On the benefits of relaxing the periodicity assumption for networked control systems over CAN,”, *Real Time Systems Symposium*, December 2009.

[15] Wikipedia. Operational amplifier applications, 2012, http://en.wikipedia.org/wiki/Operational_amplifier_applications, [Online; accessed 16-January-2012].

[16] Evidence srl., http://www.evidence.eu.com/

[17] G. Buttanzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[18] CAN Specification version 2.0. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, 1991.

[19] K.J. Åström and B. Wittenmark, *Computer controlled systems*, Prentice Hall, 1997.

[20] C. Lozoya, P. Martí, M. Velasco, J.M. Fuertes, “Analysis and design of networked control loops with synchronization at the actuation instants,” in 34th Annual Conference of IEEE Industrial Electronics, pp.2899-2904, 10-13 Nov. 2008