DESIGN OF EMBEDDED SYSTEMS ABLE TO ADAPT BOTH SCHEDULING AND CONTROL PARAMETERS

Daniel NICULAE[1], Ioan DUMITRACHE[2]

În această lucrare vom prezenta implementări ale sistemelor numerice de conducere a proceselor şi vom explica problemele ce apar datorită modului de dezvoltare independent din prisma teoriei conducerii şi a teoriei sistemelor de timp real. Vom studia apoi efectele care vor apărea în cadrul sistemului în momentul implementării acestora pe echipamente hardware nededicate controlului, într-un mediu privat de resurse suplimentare, şi modul în care aceste probleme pot fi eliminate folosind teoria de co-design.

The paper will present the implementation of numeric systems for process control and it will explain the problems that arise from the independent way of development of the actual control tasks – oriented to control theory away from the real time system- oriented to operating system theory. It will then be described the effects o running such control on an on-the-shelf resources deprived hardware environment, and the way these can be surpassed using co-designed control theory.

Keywords: co-design, process control theory, real time operating systems

1. Introduction

In today's world, process control is designed and implemented using digital devices. Although once, this was reserved to large scale system such as industrial installations and robots, today numeric control has shifted and it is more and more used in everyday devices as household equipments, automobiles and communication devices. All these demand a rapid and predictable running, in order to maintain a high level of performance, and so a high customer satisfaction.

Because the digital control system is not added to a physical system but rather embedded inside it, in the last few years the digital devices are no longer regarded as an improvement of the device but more as part of it. The term CPS – cyber physical system [1] - describes fully this type of paradigm. Thinking this way a device or even a large system is designed keeping in mind that the digital control will be an integrated part of it and so, some features will be implemented to better cope with this.

[1] Eng., SC IPA SA. Bucharest, Romania, e-mail: dniculae@gmail.com
[2] Prof., Automatic Control and System Engineering Department, Automatic Control and Computers Faculty, University POLITEHNICA of Bucharest, Romania

Control systems are forced to work in an environment where their response to the external changes must be not only accurate, but also timely. To do so they must be controlled by an entity that was designed to work in a real word environment: the "real time operating system" -RTOS. The main features that parts this operating system from others are their smaller size and more important the way they regard the correctness of a result [2]. So said, a correct real time result is not the one that has the right value but the one that also supply that value at the correct time.

Otherwise, the RTOS behave just like a regular operating system so it can be adapted to deal with various and multiple jobs. These characteristics are welcome by designers because since the control system is embedded into a product it is needed to perform as much as possible. Implementing this behaviour is somewhat easy, jobs being assigned to software entities known as task, same as any other operating system but keeping in mind that all tasks must respect the real time running paradigm. To do so, a variety of real time scheduling policies had been developed and used, according to the needs of the system some designed for real time system and others just adapted to this kind of behaviour.

In this paper the interest is focused on the way that digital system behaves as a control system so it will be perceived only from this point of view. The control theory developed the methods to implement digital control. Translated to software application these theories emerge as control tasks that need to behave deterministic. According to control theory these tasks must run at precise separated time intervals – the signals must be sampled according to a given frequency, and all the computation must take place instantly – there is no delay between reading the process values and supplying the next command. [3]

Implementing the time restriction on a task running on a real time operating system will make the design of the scheduler difficult and in some cases even impossible [4]. In most cases the theory demands are relaxed, in order to obtain a system able to run. An example of this is the implementation of control tasks on dedicated systems were the allocated resources and the design of the entire system was done in such a way that the error induced by control tasks not being executed exactly at regular intervals and the time spent between signal acquisition and the new command does not affect the performances which are expected from the system.

Lately things started to change, and given the increasing number of control system the demand for standard platform increased. With this, the behaviours of the control system also become more affected by the difference between the theory and the way system are really implemented.

A new approach emerged in order to design better system with respect to the need to implement them on cheap and reusable hardware [5]. This approach is different from the traditional one by the design of the numeric controllers by

people understanding both: control theory and real time system needs and constrains. Conventionally the numeric controllers were designed in separate stages. One stage was: dealing with the design of real time system, which had to implement all tasks and to allow all of them to run as specified. Other stage was to design the models of numeric controllers more focused to obtain the best way to control any given process. By designing the digital controller by a mixed team, the need and constrains of both teams can be better understood and systems can be change in order to achieve better global performance that by transferring the burden of implementing chances to only one part of the team [6]. Even more, systems that originally could not be created can now be implemented. This approach is known as co-design or implementation aware design.

This paper will present how the co-design theory can eliminate the negative effects of implementing control tasks on standard digital controllers:

- Chapter 2 presents the general way an ideal embedded control task is implemented
- Chapter 3 presents scheduling techniques and the errors that can appear due to schedulers.
- Chapter 4 introduces the Jitter concept, describes the jitter properties and the jitter impact on the behaviour of the control tasks.
- Chapter 5 develops the concept of adaptive digital controller. It will be discussed both the concept of tuning the control algorithms parameters in order to minimize the negative impact of jitters but also the concept of decreasing the task activation period in order to faster reject process error

2. General design of control tasks

A system control activity can be split into three basic activities: data acquisition, computing the new command, and supply the new command to the controlled process [6]. When all these are implemented in a task form, they can be implemented as a single task or as separate tasks[3].

The following are requested for the control task to be equivalent to the model described in the control theory:

- The signals acquired from the process must be read at a precise time interval t_k
- The acquisition period must be the same as the period T used to design the discrete controller.
- The new command must be computed instantly after the process data are read.
- The commands must be supplied to the process with the same period t_k

[3] The reason for this kind of implementation lays in the need of flexibility on dealing with tasks scheduling. Because it is not the purpose of this paper to analyze these matters we will consider that control tasks are implemented as a single, unitary task.

- The delay τ between the signal acquisition and new command- if such delay is present it must be constant for all instances.

Although the digital control system is created based on these concepts, most of them impose too many restrictions to be implemented alongside other tasks besides control tasks. For instance it is not possible to correctly schedule tasks with different periods [6]. Also, if the same time delay is needed the WCET (worst case execution time) can be used for a task, but doing so, it will keep the processor from doing other jobs although in some cases the current computation will be finished faster.

When implementing a digital control the following are the most used models [3]:

- The signals are read and written with a periodic pattern equal to the design period of the theoretical model. There are no delays between data acquisition and command. This is the ideal case and it is the most used model, although it imposes many restrictions and can propagates implementation errors.
- The signals are read and written with a periodic pattern equal to the design period of the theoretical model. There is a constant delay between signal acquisition and the new command. The delay is defined as: $0<\tau<t_k$. This model is used to model the delays induced by the computation time and communication channels.
- The signals are read and written with a periodic pattern equal to the design period of the theoretical model. The command is supplied at the same time with the next data acquisition. In this case the delay is equal to the sampling period $\tau=t_k$. This model can be used for an easy way to implement the controllers; everything being governed by hardware interrupts.

The figure below holds a representation of the three cases [6]:

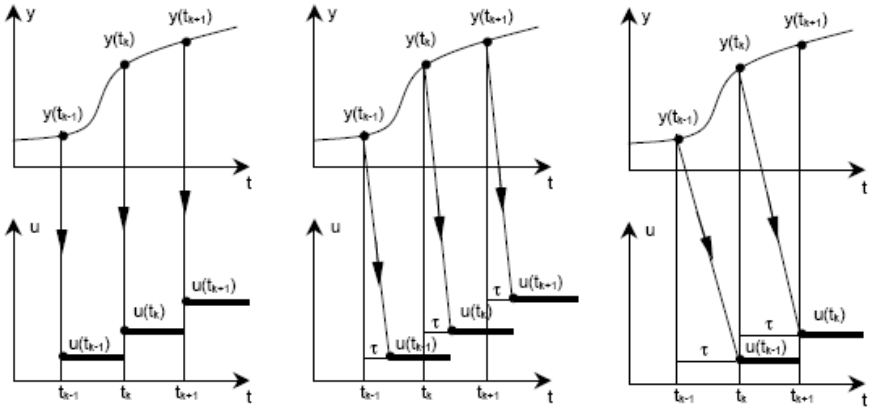

Fig. 1. Control tasks implementation

3. Scheduling process control tasks

It is important to reaffirm that control tasks will have to guarantee that at each instance they run, they will get a constant sample period for its signals. For this, the following must happen [4]:

- Each control task will be implemented as a periodic task. The period will be equal to the period of the mathematical model.
- Each control task must have a deadline equal to the one assumed when the digital controller was designed. This delay should be the same as WCET of the task, this being the best assumption for this.

Translating the time demand to task implementation parameters, the following relation can be used:

$$(T_i, C_i, D_i) \rightarrow (t_i, \tau_i, c_i) \qquad (1)$$

Where T is the model period, C (or WCET) is the execution time, D is the deadline, t is the instance period, τ is the instance delay and c is the instance execution time.

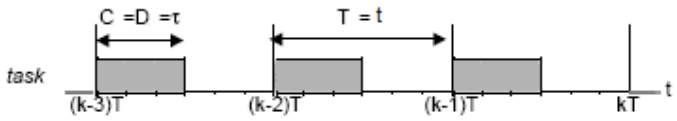

Fig. 2.Tasks instances and time parameters

Consider now an example of a digital controller which holds two tasks. For this system the design of task scheduler that can meet the control theory time demand.

Table 1

Examples1: tasks parameters

	Ti	Ci	Di
Task1	3	1	1
Task2	4	1	1

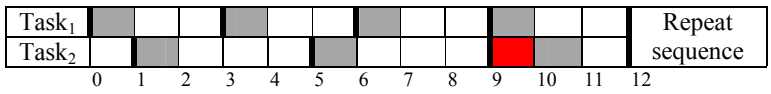

Fig. 3 Example: control task set that cannot be scheduled to respect time execution parameters

This example showed that it is not possible to schedule the entire task set and still comply with the control system time demands. For instance, the second

task is delayed one time interval at 9th time slot because $task_1$ was running at that time[4].

Since last decade, [9] it had been stated that cyclic tasks can be scheduled only if there is a harmonic relation. Two scenarios can be described:

a. Given a task set defined by the periods $T_1, T_2, \ldots, T_n$ (T_i equals the sampling period chosen when the controller was designed), there is a strict harmonic relation if following relation exist:

$$T_{i+1} = kT_i / k \in N, \forall i < n-1 \quad (2)$$

The following is an example for the harmonic relation:

Table 2

Example: harmonic task set

	T_i	C_i	D_i
$Task_1$	3	1	1
$Task_2$	6	2	2

Fig. 4. Example: harmonic task set scheduled

b. Given a task set defined by the periods $T_1, T_2, \ldots, T_n$ (T_i equals the sampling period chosen when the controller was designed), there is weak harmonic relation if:

$$T_i = k_i T_1 / k_i \in N, \forall i < n$$

The following is an example for the week harmonic relation:

Table 3

Example: week harmonic task set

	T_i	C_i	D_i
$Task_1$	4	1	1
$Task_2$	6	1	1

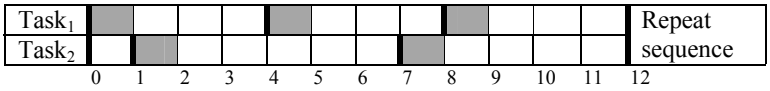

Fig. 5. Example: week harmonic task set scheduled

[4] It is up to us which task we delay.
[5] It is clear that a 6 time unit interval would had been sufficient in this case

It must be added that for the weak harmonic relations it is not always possible to have a schedule that will meet all the time demands. One example is choosing C_i=2 and D_i=C_i, inside the previous task structure.

4. Jitters

The following example is introduced to better explain jitters: of a set of two tasks scheduled [5] according to **Fig. 6**:

Table 4

Example: task set

	t_i	C
$Task_2$	7	2
$Task_1$	4	2

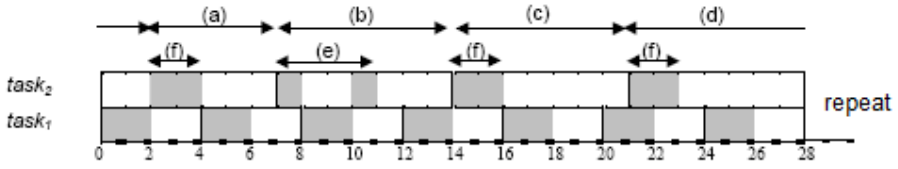

Fig. 6.Example: scheduled task period

The two tasks presented have a totally different behaviour. The first task has a periodic execution and a constant execution time. The second task has four different execution intervals and it has two different execution times. If this task will be executed as a control task using the general digital theory, it will generate a faulty system.

Jitters will be defined as any time interval that affects the timely execution of a task. Jitters are present in systems where high priority tasks are:

c. stopping the already working low priority tasks.

d. delay the start of low priority tasks.

According to this the following definition will be made:

- sampling jitter: any time interval that delay the task from starting at the designated time:

$$\Delta t_e = \{\forall \Delta t / start(task_j) = jT + \Delta t\} \quad (3)$$

For systems presenting sampling jitters the sampling interval for a instance is defined as a non periodic interval t_{ij} = start(i,j+1)-

start(i,j)≤T_i. (i represents the task number and j represents the instance of the i^{th} task)

- sampling actuation jitter: any time interval that delays the end of a started task

$$\Delta t_a = \{\forall \Delta t / stop(task_j) = jT + C + \Delta t\} \quad (4)$$

For systems presenting sampling actuation jitters the actuation delay is defined as: τ_{ij}=stop(i,j)- start(i,j), and τ is not a constant.

Returning to the example above it can be said that the second task is affected by:

- sampling jitter during the first, second and fourth instance, and only the third instance is jitter free.
- sampling actuation jitter during the second instance.

Jitters have a negative influence over the control system [5]. Implementing control tasks without regarding jitters can sometime greatly decrease control performances and even bring instability to an otherwise stable system.

Using a inverted pendulum as an example and the control task being scheduled as the less important task in a non pre-emptive system the following results:

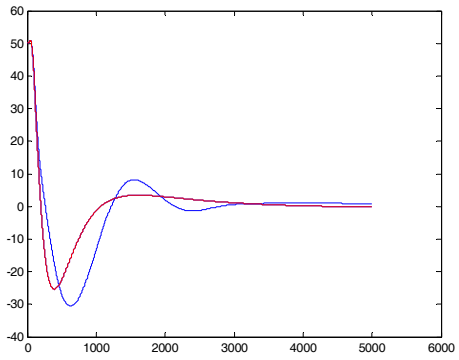

Fig. 7. Example: process controlled by a control task ideal case (red), jitter affected (blue)

where the blue line represents the behaviour of the system affected by sampling jitters and the red line represents the behaviour of an ideal system. Both simulations had been made in the same initial conditions and same controller's parameters.

5. Implementation aware design

The previous section presented that, implementing the controllers by a regular method on a jitter plagued system will decrease system performance. The causes that make the jitter appear were also presented.

Keeping this in mind a new approach was created to better design control system by making a compromise between the needs of control systems and real time system [6]. In this way the control systems will constantly readjusting controller parameters to cope with each situation induced by jitters and will set its time requirements so that the operating system can schedule all tasks properly. The operating system will, in change, create the framework, so that control tasks will be less affected by jitters.

In order for this to happen, a mechanism is implemented to transfer to the control tasks, the data about the state of the task status such as sampling interval used by the next instance and the most probable execution time if not the exact next execution time.

The control task will be also changed to cope with this approach and will execute a changed code. An example for a changed state controller is presented below. The controller is on line calculated at every task instance:

Classic controler	Implementation aware controler
{ [y_k, ,w_k] = read imputs; u_k = compute_ command(x_k,..., w_k, -F); x_{k+1} = compute_state(x_k, $u_{k....}$,, Φ, Γ) return (u_k) }	{ [z_k, w_k] = read imputs; [t, τ] = get temp data; [Φ(t,τ), Γ(t,τ)]= compute_model (A , B, t, τ) F_k = compute_controler (Φ(t,τ), Γ(t,τ)) u_k = compute_ command (x_k,,w_k, -F); x_{k+1} = compute_state (x_k, u_k,, Φ(h,τ), Γ(h,τ)) return (u_k) }

There are three great approaches to this:

Designed imposed

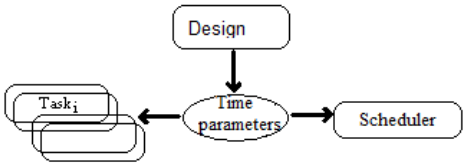

Fig. 8. Co-design, 1st case

The time parameters are set from the design stage and the entire digital controller is run accordingly. This approach is used mainly for the fixed schedule

policy where the system is completely designed and at the run stage it will only execute schedule instruction form a cyclic list.

The control tasks will change their parameters accordingly.

Scheduler imposed

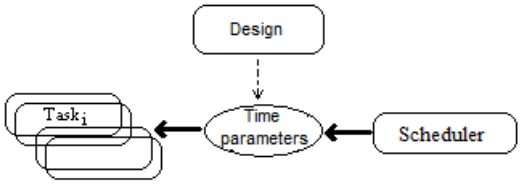

Fig. 9. Co-design, 2nd case

During the design stage a set of parameters are „suggested" to be used by the system. The parameters used during the system running time are changed according to the needs of the operating system. This approach assures that dynamic tasks will be run on the system and also that control tasks will receive the complete information to adjust their parameters.

Controller imposed

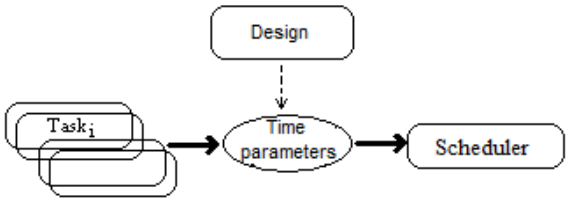

Fig. 10. Co-design, 3rd case

The main idea behind this approach is that control tasks can access spare computing resources whenever they need. The system still receive a set of parameters at the design stage, parameters that are regarded as the stable state parameters and that are used when controlled processes have a low error. The scheduler will still dispatch the tasks according to its algorithms so that jitters will still be present and of course compensated.

The control tasks can demand the increase[6] of access by decreasing their assigned execution interval. The resource will be freed if other task has a greater need or if it is not needed by the current owner.

An example of what had been presented above can be shown using an inverted pendulum as a control process. The process will be control by a dynamic scheduled digital system where a set of tasks will run. The set of control tasks will have a total of 20ms, our task being the first to run and having a execution time of 1ms.

Table 5

Example: task set

set	T(ms)	D(ms)
$Task_{ctr}$	100	20 (1)
$Task_1$	60	20
$Task_2$	70	20

The digital system has the following schedule behaviour:

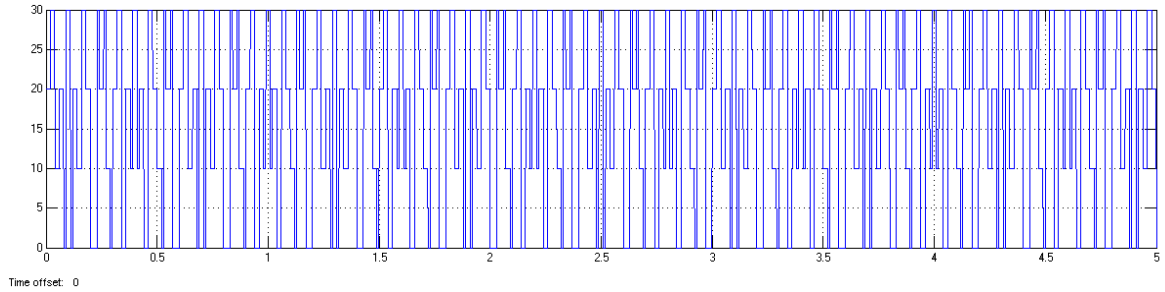

Fig. 11. System scheduler for the task set using the RM algoritm

The system's answer using jitter affected controllers is:

[6] According to control theory, a controller behaves better to reject an error if it has a higher frequency rate.

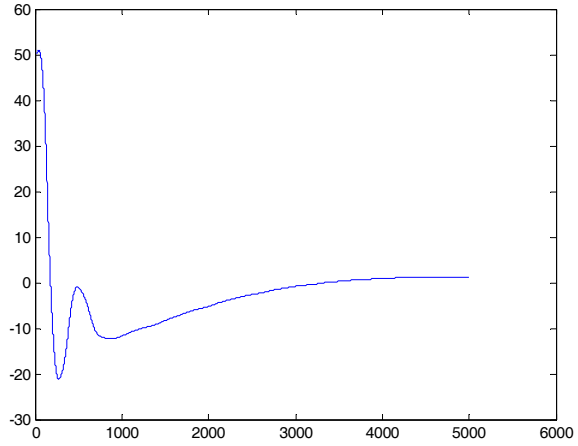

Fig. 12. Inverted pendulum response controlled by a jitter affected control task

It can be seen that although the inverted pendulum does not return smoothly to its upright position at the end a zero error is achieved. If the controller is implemented using a modified version, the new behaviour is:

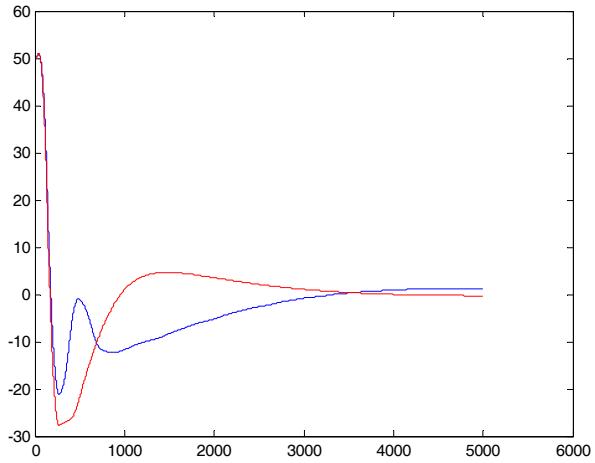

Fig. 13. Inverted pendulum response, control by a jitter afected and a modified control task

It can be seen that the overall system behaviour is much better, the system reaching much faster the equilibrium. This situation can be assimilated to both the first and second approach - the controller receives a set of time parameters but the real time operating system schedules the task with jitter.

At last, it is presented a controller that demands the increase of its frequency execution rate to cope with increase error:

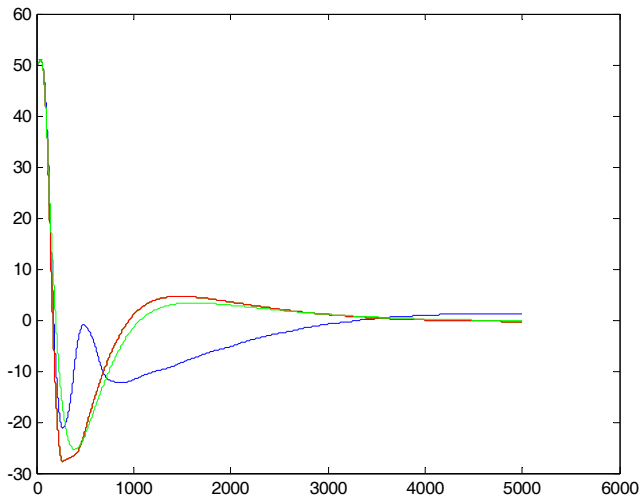

Fig. 14. Inverted pendulum response, control by a jitter afected, a modified control task and a control task with dynamic change of execution rate

This last behaviour is better still than the one above.

To give a numeric form to our simulations the sum of absolute error for the entire length of the simulation was computed (5s).

Table 6

Sum of the absolute process error during simulation time

	Classic controller	Modified controller	Dynamic controller
System error	2.7667	2.5145	2.3377

6. Conclusions

The implementation aware design for control system is studied for some time but only in the last decade it started to be considered a noticeable domain. Studies had been directed first to controller task implementation and then towards scheduler design.

Usually this type of implementation is targeted to high usage level processor, in order to better allocate computing resources. Reviewing the design technology it can clearly be seen that the implementation aware design for control tasks add extra computing time not only to the last added task but possible also to other control tasks running on that operating system. According to the above, it is

not always possible to implement this type of design. Using the same reasoning it can safely be said that before implementing such a system, an extensive study of the time parameters inside a running controller must be carried to determine if the advantage offered by this approach will overcome the increase in complexity induced.

Another issue raised by this theory is the complexity of the scheduler. For the same reason as above, a complex scheduler can also induce unnecessary computer time. Many scientific papers propose complex solutions to obtain optimal response in regard to one or more of the process control tasks.

Further studies must be directed towards optimizing by reducing the added system complexity in order to obtain implementable results.

R E F E R E N C E S

[1] *P. Marwedel,* Embedded system design, Embedded system foundation of cyber-physical system 2nd edition, ISBN 978-94-007-0256-1, 2011

[2] *M. Dragoicea*, Programarea aplicatiilor in timp-real. Teorie si practica (Real time application programing, Theory and applications), Editura Universitara, ISBN 978-973-749-579-2, 2009

[3] *I. Dumitrache,* Ingineria Reglarii Automate (Automatic control system), Romania 2005

[4] Scheduling Theory, Algorithms, and Systems. Third edition Michael L. Pinedo, ISBN 978-0-387-78934-7 Prentice Hall, 2008

[5] *C. Aubrun, D. Simon, Y. Song*, Co-design approaches for dependable networked control systems; ISBN 978-1-84821-176-6, 2010

[6] *M. Lluesma, A. Cervin,* Jitter Evaluation of Real-Time Control Systems 2006

[7] *M. Ben Gaid, A. Cela, Y. Hamam, C. Ionete,* Optimal Schedueling of Control Tasks with State Feedback Resource Allocation, American Control Conference ACC'06, Minneapolis, USA June 2006

[8] *P. Marti, R.Villa, J.M. Fuertes, G. Fohler,* A Controller Design Method to Compensate for Real-time Scheduling Inherent Jitters. Submitted to the 41th IEEE Conference on Decision and Control, Las Vegas, USA, December 2002

[9] *P. Marti*, Control Performance Evaluation of Selected Methods of Feedback Scheduling of Real-time Control Tasks, 2008