

## MODELLING SELF-TRIGGERED TASKS FOR REAL-TIME CONTROL SYSTEMS

**Manel Velasco, Pau Martí, Josep M<sup>o</sup> Fuertes.**

*Automatic Control and Computer Engineering Department. Technical University of Catalonia,  
Barcelona, Spain.*

*Tel: +34 93 401 79 74 / Fax: +34 93 401 70 45  
{manel.velasco, pau.marti, josep.m.fuertes}@upc.es*

**Abstract:** In real-time control systems, the objective of control activities, i.e., to control processes, and the objective of scheduling techniques, i.e., to meet deadlines, are accomplished separately. This may derive in sub-optimal designs in terms of both control performance and resource utilization. Control activities optimise control performance regardless the computational demands of other tasks and scheduling techniques optimise the use of resources regardless the dynamics of the control application. To overcome this problem, we present a control-based model for control tasks in which computing resources and control performance are jointly considered. Concretely, the model allows each control task to trigger itself: at each control task instance execution, the executing instance informs the scheduler when the next instance should be executed. The next instance execution point in time is dynamically obtained as a function of the utilization factor and control performance. Preliminary results show that control activities, at run time, are able to define self-execution patterns that dynamically balance optimal levels of control performance and resource utilization.

**Keywords:** Real-Time Systems, Task Models, Control Systems, Nonlinear Discrete-Time Control Systems.

## 1. INTRODUCTION

In real-time control systems, the objective of control activities, i.e., to control processes, and the objective of scheduling techniques, i.e., to meet deadlines, are accomplished separately. This may derive in sub-optimal designs in terms of both control performance and resource utilization.

On one hand, control activities optimise control performance regardless the computational demands of other tasks. This fact comes from the control design process: a discrete time controller is designed assuming a constant sampling period. In terms of task execution, that means that at run time the controller will execute demanding a constant processing capacity. Therefore, in the design process of the controller, it is not usually taken into account the possibility of taking advantage of processing capacity that may be released by other tasks. That is, the controller design does not allow increasing the execution rate (decrease the task period) to exploit available resources.

On the other hand, scheduling techniques optimise the use of resources regardless the dynamics of the control application. For instance, a periodic control tasks may not require the designed execution rate (the assigned processing capacity) if the controlled plant is in equilibrium. When a plant is in equilibrium, after an execution of the controller, no major change in the state of the plant can be appreciated. That is, the contribution of the controller execution can be considered as useless. Therefore, the processing capacity has been used when not necessary. In such situations, this processing capacity could have been used by other tasks with higher processing capacity demands.

To overcome these problems, we present a control-based model for control tasks in which computing resources and control performance are jointly considered. Concretely, the model allows each control task to trigger itself: at each control task instance execution, the executing instance informs the scheduler when the next instance should be executed. The next instance execution point in time is dynamically obtained as a function of the utilization factor (global parameter) and control performance (local parameter)<sup>1</sup>. Therefore, task-timing constraints (e.g., task period) are dynamically adjusted.

Consequently, we could say that each control task acts as a co-scheduler, helping the scheduler at the scheduling decisions. Note that scheduling decisions will depend not only on task deadlines (as standard scheduling policies do) but also on the utilization

factor and the dynamics of the control applications. Figure 1 illustrates the operation of the whole system.

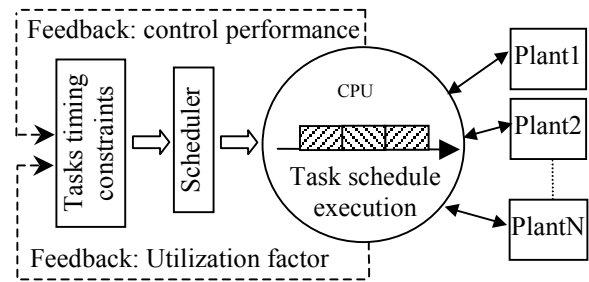


Figure 1. System operation model

Preliminary results show that control activities, at run time, are able to define self-execution patterns that dynamically balance optimal levels of control performance and resource utilization.

The rest of the paper is organized as follows. In section 2 we discuss the state of the art. In section 3 we provide basic background to define the problem formulated in section 4. In section 5 the self-triggered tasks model is developed. Preliminary simulations results are presented in section 6. Finally, in section 7 we conclude and point to future research work.

## 2. STATE OF THE ART

In this section we present a brief but relevant discussion on the state of the art. The model we present resembles the model presented in [CER02]. They propose to use feedback information from the controlled plants to take scheduling decisions. Specifically, all control tasks periods are proportionally enlarged or shorted at a given time instant as a function of the utilization factor. Therefore, they do not allow the exchange of processing capacity if the control application requires higher execution rates of specific tasks.

The later can be achieved using the elastic model of [BUT02]. In such model, the elastic coefficient of each task allows the scheduler change the task execution rate within specified ranges. The elastic coefficients are regarded as fixed parameters to be specified before run-time. The model we propose matches the elastic model if each elastic coefficient of control task would be treated as a dynamic parameter, being a function of the resource utilization and control performance.

It should be stressed that the work we present derives from [MAR02]. The authors point out that novel methods for control task scheduling in which scheduling decisions should depend on control performance and resource utilization are needed.

Some similarities may be identified between our model and event-based systems. In event-based systems the sampling period takes random values. The sampling period for our model is also variable, but there exist a slightly difference: in event-based

<sup>1</sup> The utilization factor of the system is obtained taken into account all tasks in the system. Therefore, it is a global parameter and affects all tasks. The control performance is obtained by each task from the corresponding controlled plant. Therefore, it is a local parameter and affects itself.

systems the next execution point in time is unknown; the suggested model in this paper deals with known future periods because they are a result of the model execution.

### 3. BACKGROUND

Traditionally, real-time tasks are characterised by the period ( $T_i$ ) and the worst-case execution time ( $C_i$ ) [BUT97]. For control tasks, the task period is given by the sampling period ( $h_i$ ) that, among other parameters, determines the performance of the controlled system.

The sampling period  $h_i$  can be selected from a range of values. It has a lower limit (the shortest sampling period), which is given by the underlying technology (processing power). The shortest sampling period should include the worst-case execution time of the controller. This is represented in Figure 2 a) where all instances of a control task have as a task period equal to the worst-case execution time. However, other patterns of execution could be represented in Figure 2 by sequences like b) and c), where the task period is greater than the worst-case execution time (in Figure 2, boxes represent consecutive execution of instances of a task assuming worst-case execution times,  $C_i$ , and consecutive vertical dotted lines represent the task period).

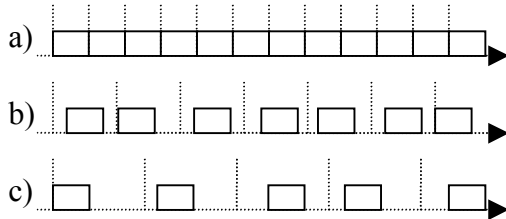


Fig.2. Possible periods for a control task. a) lower limit (shortest period), b) intermediate period and c) long period

The range of values for the sampling period has also an upper limit (the longest sampling period): beyond this limit, the control task loses the control of the controlled system. That is, the control tasks execution rate is too slow compared to the system dynamic, thus losing relevant information as illustrated in figure 3, where the effects of a low sampling rate are reflected. The dotted line shows the estimated sampled signal while the continuous line is the real signal. If the control task works with the dotted line, the information that the control task has is not accurate enough.

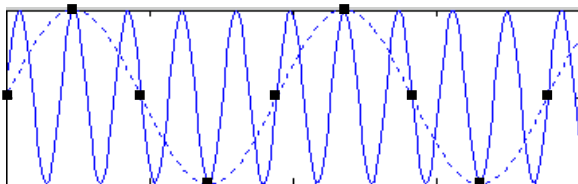


Fig.3. Real signal (continuous) and the estimation of the sampled signal (dotted). The little squares indicate the sample points.

More specifically, this upper limit depends on the natural frequency of the system to be controlled. This frequency can be easily found in different ways. For instance, the natural frequency of oscillation for a spring is determined by the constant of the spring ( $K$ ) and the existent mass at his end ( $M$ ). It also may be determined in an empiric way, stretching the spring lightly and observing the oscillation frequency of the system. The Shannon's theorem (see for example [PHI84]) tells us that the longest sampling period (the upper limit) should be at least given by (1), where  $f_0$  denotes natural frequency.

$$h_i = \frac{1}{2f_0} \quad (1)$$

To avoid problems derived from working on limit situations, the recommended sampling period is usually taken following well-known rules-of-thumb (see for example [ÅST88][ PHI84]). For example, a rule-of-thumb suggests taking a sampling period from 4 up to 20 times of theoretician one, as shown in expression (2)

$$h_i = \frac{1}{8f_0} \text{ to } \frac{1}{40f_0} \quad (2)$$

Each possible choice for the sampling period has advantages and disadvantages. In short, a short period allows a quick reaction in front of perturbations (which is positive from a control point of view), but increases the processor's load (which is negative from a resource utilization point of view). Using long periods decreases the processor's load but may give poor response in front of perturbations.

COMFORMATO

Note that the choice of the sampling period has to balance the desired control performance and the feasible computational demand. INCRUSTAR

### 4. PROBLEM FORMULATION

The selection of the execution period for a control task has been discussed in the previous section. The control engineer prefers small task periods in order to obtain good responses of the controlled system. The real-time engineer prefers long task periods to relax specifications and to facilitate task set schedulability. Both preferences are in conflict, and traditionally, a single value for the task period has to be choice before run-time [MAR01].

If we look at control tasks operation, we can distinguish two clearly differentiated situations that affect the execution rate of a control task. Firstly, we have the situation in which the controlled system is clearly outside of the desired working point. That is, the controlled system behaves differently as it should do. In this situation, a small period for the control task would imply a fast correction of the non-desired behaviour of the controlled system, bringing the system to the desired working point. Once the desired point is reached, no more correction is needed. Looking at control signals, this means that control actions tend quickly to zero. If control actions are

almost zero, a small period is no longer needed because the contribution of each control action has no effect on the controlled system (they are zero).

The opposed situation appears when the system is in the desired working point (in equilibrium). In such situation, as be stated previously, the period of the control tasks can be long because the contribution of control actions have no effect on the controlled system (they are zero). However, a perturbation may suddenly affects the controlled system, bringing the system away from the desired working point. Then, if the task period is large, the controlled system will respond slowly, taking long time before reaching again the equilibrium point, which may not be good for the given performance specifications.

In fact these two situations, which dynamically appear due to perturbations that affect the controlled system, can be considered as “transitory”. Generally, the transition from one to the other can be considered as a soft evolution.

Note that from a control point of view, depending on the status of the system (depending on the situation that the controlled system presents), if the control tasks has a constant value for the task period, the real profit of the control task CPU cycles (given by the task period) can be very high or very low. When the controlled system is in equilibrium, from a control point of view, the CPU cycles of the control task are not fully exploited, thus wasting resources. In the opposite situation, the CPU cycles are appropriately exploited.

Concluding, long periods for control tasks are preferable when the controlled system is stable in the desired working point and small ones when the controlled system is far away from the desired working point. This demands models that can dynamically accommodate different values for the task (controller) period. However, traditional control and scheduling techniques do not provide this feature.

The model we present allow control tasks to have varying values for the period. The exact value for the period (at each control task instance execution) is dynamically adjusted depending on the controlled system status and the CPU load. Figure 4 shows the relation between the task period and the controlled system status. The curve at the bottom of the figure represents the status of the controlled system and the curve on the top of the figure represents the value for the task period for each instance execution. When a perturbation appears (when the bottom line presents a “mountain” shape) and the controlled system is brought away from the desired working point, the task periods are decreased. When the controlled system response reaches the desired working point (it has a horizontal shape) the task period is adjusted to the nominal value (longest period).

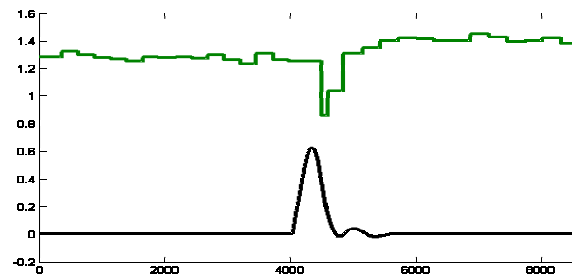


Fig.4. System evolution while period adjusting. Upper line corresponds to periods, lower line corresponds to system.

## 5. SELF-TRIGGERED TASK MODEL

The main idea of the model we present is to use the common models that are used in the analysis and design of control systems. Concretely, we propose to use an extended state-space representation. State-space models allow us to describe the future response of a system, given the present state (characterized by the state variables), the excitation inputs and the equations describing its dynamics (see [ÅST88] o [PHI84] for further reading on discrete-time state space models). The extension we suggest is to incorporate as new state variables the task period and the utilization factor. Therefore, we will be mixing the control behaviour (already represented in the state space model) with the execution rate of the task and the processing demand. In the following subsections, step-by-step, we develop the model.

### 5.1. Basic model

Let us think on a closed loop system formed by a ball and beam (see figure 5), which is the plant to be controlled, and a control task that has to be executed on a processor and has to control the ball and beam. The ball and beam system has a motor that balances (by rotating movements) a beam in order to keep the ball (that can rotate freely along the beam) in the desired beam position [AST88], as illustrated in figure 5..

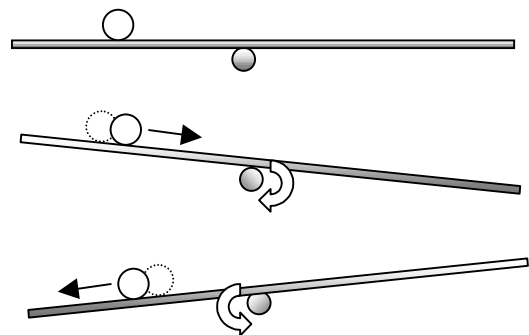


Fig.5 Ball and beam system.

The objective of the controller is to actuate on the motor to locate the ball in the desired position. To do so, at each sampling time, the controller takes the value of the position of the ball and the angle of the beam and generates the new angle for the beam that derives in the corresponding actuation on the motor.

The ball and beam system is habitually represented by the linear discrete-time invariant state-space model [AST88] given in (3)

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} \frac{h^2}{2} \\ 2 \cdot h \end{bmatrix} U \quad (3)$$

In equation (3),  $x_k$  and  $y_k$  (which are the state variables) represent the position of the ball and the beam angle respectively at the  $k$  sampling instant. The first matrix (2x2 dimension), called system matrix or state transition matrix, describes the dynamics of the ball and beam. The second matrix (2x1 dimension), called input coefficient matrix, links the input with the system dynamics. In both matrices,  $h$  is the sampling period.  $U$  is the available vector of inputs; in our case it is the tension (1x1 dimension) that we provide to the motor. The input can adopt positive and negative values, allowing the motor of the beam to rotate to both sides.

To control the system it is necessary to use a rule (a control law) that allows us to find values for  $U$  at each sampling instant that guaranties the desired behaviour for the system. By means of classical control techniques it is possible to design a control law that generates appropriate inputs  $U$  to the system in order to locate the ball at the desired position.

Note that in the state space representation of the ball and beam, the variables that describe the system are  $x_k$ , the angular position of the beam (angle), and  $y_k$ , the position of the ball on the beam. At each sampling instant,  $x_k$  and  $y_k$  vary according to the system dynamics (state transition matrix) and the input (applied through the input matrix). However,  $h$ , the sampling period, which appears on the matrices as a result of the discretization process, has a constant value that has been chosen at the controller design stage. Recall that (3) is a discrete-time model obtained via discretization of the continuous-time model. Therefore,  $h$  has nothing to do with the state of the system, although it influences its dynamics.

## 5.2. First model modification.

Up to now, we have described the classical ball and beam state-space representation. As we stressed in section 3, we want a model able to accommodate different values for the sampling period (i.e., the task period) according to the desired control performance and taking also into account the processor load.

The first extension for the previous model allows us to have varying sampling periods according the controlled system dynamics. To achieve this objective we extend the state representation of the ball and beam system model with a new system variable, the task period,  $h_k$ . Therefore, at each task instance execution, the task period will be changed according to the state of the system given by  $x_k$ ,  $y_k$  and the new state variable  $h_k$ . This is intuitively expressed in (4)

$$h_{k+1} = h(x_k, y_k, h_k) \quad (4)$$

Recall that the state of the system can be directly related to control performance. For instance, a simple rule could be *the smaller the norm of the state vector, the better the controlled system performance*. In terms of the ball and beam: the smaller the deviation of the beam with respect to the horizontal position and the smaller the distance of the ball with respect to the desired location, the better the performance. Therefore, at each task instance execution, the added state variable determines the next task instance execution point in time as a function of control performance.

It has to be pointed out that if the system state is increased by the new variable, a new control law giving the appropriate sequence of values for the input  $U$  is needed. The extended model is represented in (5)<sup>2</sup>.

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h_{k+1} & 0 \\ 0 & 1 & 0 \\ \alpha & \beta & \omega \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \\ h_k \end{bmatrix} + \begin{bmatrix} \frac{h_{k+1}^2}{2} \\ 2h_{k+1} \\ 0 \end{bmatrix} \cdot U \quad (5)$$

Note that in the system given by (5),  $h_k$  is the new state variable. The dependency of this variable with the others system variables is given by parameters  $\alpha$ ,  $\beta$  and  $\omega$ . Let's discuss some properties of the extended model, depending on values of  $\alpha$ ,  $\beta$  and  $\omega$ :

- If  $\alpha$  and  $\beta$  are set to zero and  $\omega$  is set equal to 1, then, for each  $k$ ,  $h_{k+1} = h_k$ , and the system may be considered as the original one (given in (3)). The control law that will give the sequence of inputs  $U$  can be obtained by classical controller design methods. This model will result on the classical real-time implementation with the task period equal to the sampling period ( $T=h$ ).
- If  $\alpha$  and  $\beta$  are set to zero and  $0 < \omega < 1$ , the sampling period will be decreased at each instance execution, tending to 0, thus leading to a non real system. From a schedulability point of view, at some point in time, the task period will be shorter than the task worst-case execution time, leading to an unfeasible system.

<sup>2</sup> Note that  $h_{k+1}$  (and not  $h_k$ ) appears inside of the system and input matrices. This is due to the solution of the system equations. In the non-extended model, the sampling period of the system and input matrices has no index ( $k+1$  or  $k$ ) because it is constant ( $h$  at the  $k$  instant and  $h$  at the  $k+1$  instant have the same value). In the present model, since  $h_k$  is a system variable that varies form instance execution to instance execution, it is necessary to distinguish in the matrices which is the appropriate  $k$  index.

- If  $\alpha$  and  $\beta$  are set to zero and  $\omega \geq 1$ , the sampling period will be increased at each instance execution, tending to  $\infty$ , thus violating the Shannon's sampling theorem. Note that from a real-time point of view, this tendency could be desirable because implies less pressure on the use of resources.
- If  $\alpha$  and  $\beta$  are set different from zero and  $0 < \omega < 1$ , we have a system with a variable period, each one depending on the previous system state. The computed value for each  $h$  is given by (6).

$$h_{k+1} = \alpha \cdot x_k + \beta \cdot y_k + \omega \cdot h_k \quad (6)$$

The formula for  $h_{k+1}$  shows that the new value for the next task period depends on the previous one ( $\omega h_k$ ). This implies smooth transitions in period variations. This model will require real-time implementations able to accommodate varying task periods (which depend on the state variables including the previous period), similar, for example, to the models used in [BUT02]. The main problem of this combination of values for  $\alpha$ ,  $\beta$  and  $\omega$  is that the state space model becomes nonlinear (that is, a small change in the inputs may result in chaotic outputs), as we outline later in this section. Therefore, finding the adequate control law giving the appropriate sequence of inputs  $U$  will be a more difficult task (see for example [ISI89]), if feasible.

- If  $\alpha$  and  $\beta$  are set different from zero and  $\omega$  is set to zero, we get a variable period system depending only on the original state variables. Consequently, the more quickly these variables move (angle and position), the faster the period changes, thus loosing the smooth transitions found in the previous case. This may result in values for the sampling periods out of the permissible ranges (that we explained in section 2), which from a control point of view may violate the limit given by Shannon and from a real-time point of view, it may result in an unfeasible schedule (if the period is shorter than the worst-case execution time).
- If  $\alpha$  and  $\beta$  are set different from zero and  $\omega \geq 1$ , the evolution of the system will depend on the chosen values, which require a deeper analysis, out of the scope of this paper.

From the extended model given by (5), two elements should be highlighted:

- The system is nonlinear, since  $h$  is a state variable and it also appears multiplying to other state variables.
- It would be possible to obtain negative values for the task period from the actual extended state space model. Considering only a theoretical

view, this possibility means that the system should return to the past in order to modify already taken decisions. But this is clearly non-programmable in a real system. We could solve this problem by using the absolute value for the  $h$  at each task instance execution. This will guarantee that  $h$  will be always positive.

Taking into account the previous points, the model should be modified to the expression given by (7).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ |h_{k+1}| \end{bmatrix} = \begin{bmatrix} 1 & |h_{k+1}| & 0 \\ 0 & 1 & 0 \\ \alpha & \beta & \omega \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \\ h_k \end{bmatrix} + \begin{bmatrix} \frac{h_{k+1}^2}{2} \\ 2|h_{k+1}| \\ 0 \end{bmatrix} \cdot U \quad (7)$$

In (7),  $|h_{k+1}|$  means absolute value of  $h_{k+1}$ . Note that in this expression,  $h$  always is positive, due to the absolute value operator. Note also that  $h$  in the  $k$  instant does not have the absolute value operator because it comes from the previous task instance execution.

Finally, is also have to be stressed that the system matrix has  $h_{k+1}$  at the  $k$  instant, which is an inconsistency. However, this can be easily solved by substituting the  $h_{k+1}$  value for the expression given in (6), which is already known at the  $k$  instant.

### 5.3. Second model modification

Looking at the final model given by (7), two difficulties, beyond having a nonlinear model, can be identified:

- the absolute value operator makes the mathematical tractability implies using two symmetric models, one for positive values of  $h$  and the other for negative values. From a programmable point of view, this involves no major problems (a if-then-else structure is required). However, from a control tractability viewpoint, this model duplicity is not desirable because it requires using specific control methods like switching mode controllers (see [LEI03] for a benchmark study of switching techniques).
- The possible values that  $h$  may take are not bounded, due to the linear relation between  $h$  and the original state variables. Note that if the state variables take huge values,  $h$  will rapidly increase (and viceversa). As we outlined in the previous section, this introduces control and real-time problems.

To solve the previous problems, we suggest to bound the possible  $h$  values by introducing an appropriate function of the state variables instead of having a simple linear relation. We call this function *h-function*.

Taking advantage of the use of the h-function, we incorporate the utilization factor in the model, as a measure of the processing capacity. Recall that up to now, the model only related the varying period of the task with the original state variables (as a measure of control performance). In this new extension of the model (adding the h-function), we will relate the period variation to the control performance as well as to the processing capacity.

The second extension we present is given by (8)

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + y_k \cdot f(x_k, y_k, h_k, \zeta) \\ y_k \\ f(x_k, y_k, h_k, \zeta) \end{bmatrix} + \begin{bmatrix} \frac{f(x_k, y_k, h_k, \zeta)^2}{2} \\ 2f(x_k, y_k, h_k, \zeta) \\ 0 \end{bmatrix} \cdot U(8)$$

In (8),  $\zeta$  represents the utilization factor of the processor at the k instant and the h-function is given by  $f(\cdot)$ . This new model is obtained as a natural extension of the previous one. In the previous section  $h_{k+1}$  was obtained as lineal combination of the other state variables. In the new extension  $h_{k+1}$  is obtained by an appropriate function of the state of the controlled system and the CPU load. Note that the goal of the h-function is to allow the task period to take values from a bounded range.

This allows a grater variety of possibilities in the selection of how h changes at each moment. For instance, the same system with two different h-functions will result in very different behaviours in terms of CPU load and control performance. Note that choosing a specific h-function could facilitate system schedulability (this could be done either offline or even online) as well as improve control performance.

It is important to point out that for the state space model given by (8), the analysis and design of a control law can be a complex task. However, it is possible to design control laws that guarantee the complete stability of the system around a desired working point. These techniques range from the system linealization [ISI89] to the complex techniques of feedback linealization for discrete systems [NIJ90].

#### 5.4. The selection of the h-function

As we outlined before, the selection of the h-function determines controlled system performance and CPU load. Therefore, it is a crucial design choice. The most natural way for selecting the h-function is to mathematically translate in a function the following desired rule: as the system gets closer to the desired working point (equilibrium), the period should be as larger as possible, keeping Shannon limit (recall discussion of section 3). If a perturbation appears on the system, bringing it away from the equilibrium

point, the period should be decreased (to improve control performance) taking into account the available processing capacity. Mathematically, this can be accomplished by the h-function given by (9)

$$h_{k+1} = f(x_k, y_k, h_k, \zeta_k) = e^{-(x_k^2 + y_k^2)} g(\zeta_k) \quad (9)$$

Note that (9) has a negative exponential shape, with two main parts. The first part is the exponential function, which contributes to the h values taking only into account, measures of control performance. It allows each value to smoothly vary from an upper limit to a lower limit, if the exponent of the exponential function is kept positive. That's why we suggest putting the square exponents over  $x_k$  and  $y_k$  to convert the possible negative values (of the state variables) into positive ones. Note that as we explained in section 5.2., in this case, the state variables already are measure of control performance. Otherwise, the exponent should include the operation needed to measure the controlled system performance. The second part, the function  $g(\zeta_k)$ , allows to correct the next value of h taking into account the processor's utilization factor.

The fig.6 shows possible ranges of h values obtained using the h-function given by (9). The figure has two degrees of freedom: the control performance (horizontal axis) and the utilization factor (which would correspond to a perpendicular axis). The results are given according to the vertical axis, which are the possible values for the task period. When the control performance (understood as the system deviation with respect to the equilibrium point) decreases (the deviation increases), the values for the task period tend to short values, with the aim of quickly correct the deviation. In addition these values are increased as the load increases.

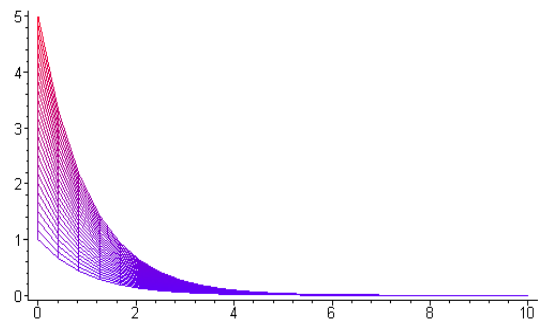


Fig.6. Possible values for the task period

#### 5.5. Concluding remarks

The use of tasks that are driven by the model given by (8) have a remarkable advantage:

*The task observes the state of the system which is composed by the processor and the controlled system.*

In this way, at each control task instance execution, the period selection is in consonance with all the elements that are involved in the control of the system

and in the scheduling to the task set, thus facilitating the optimisation of the whole system, in terms of both control performance and resource utilization.

The main goal of the presented model is that if several tasks are driven according to this model, the processing capacity can be dynamically balanced among them according to the controlled performance measured (see simulation results section). Moreover, note that although the processing capacity is dynamically exchanged, the feasibility of the task set is kept. If more tasks are added to the system, the probability of keeping a feasible schedule is high due to the fact that decisions (on the period selection) are based on the utilization factor.

## 6. SIMULATION RESULTS

In figure 6 we show the results of two *ball and beam* control tasks executing a single processor. The control laws implemented in the two tasks have been calculated by means of linealization techniques (not detailed in this paper), according to model we have presented.

In Figure 6 we can observe 4 lines. The upper ones correspond to the sequences of values for each task period, and the lower ones correspond to the dynamics of each controlled system. The task period values take into account the utilization factor, which is injected as a simulation variable. At the beginning (left side of the figure), both systems are stable at the desired working point, so both have the same value for the execution period.

Later on a perturbation affects system 1, deviating the system away from the desired working point. This perturbation causes an immediate decrease of the task period controlling system 1 and an increase of the task period controlling system 2. Therefore, the exchange of the processing capacity among the two control tasks has started. From

The delay observed between the perturbation arrival time and the first decrease in the task period is due to two factors:

- There is always an offset among the moment in which the perturbation takes place and the moment in which the task samples the system. In the worst case, this offset could be as big as the period. However, this situation doesn't necessarily lead to worse system dynamics
- After the perturbation arrival time the error is small and the decrease in the sampling period is not very significant. One period later, the error has increased and the decrease on the sampling period becomes more remarkable.

Note that the communication between both tasks only takes place through the processor's utilization factor, which is a global parameter (see section 1 for further details).

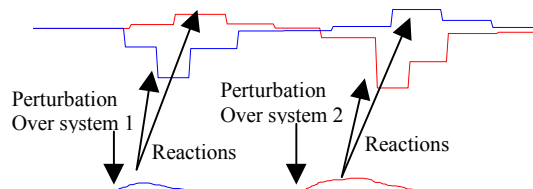


fig 7. Analysis of responses and periods enlargements.

Later on another perturbation affects system 2. The system reacts in a similar way to the previous one. That is, the processing capacity exchange takes place as before, but in inverse direction.

## 7. CONCLUSIONS

In this paper we have presented the self-triggered task model that drives control task executions according to controlled system performance and available processing capacity. Specifically, the model allows control task to adjust their execution rate, acting as a co-scheduler.

As we outlined, the main research issues behind this work is the analysis and design of the controller, which must give the appropriate inputs to drive the whole system to the desired behaviour.

## ACKNOWLEDGEMENTS

This research has received support from the Spanish Ministerio de Ciencia y Tecnologia Project ref. DPI2002-01621.

## REFERENCES

- [ÅST88] Krarl J.Åström and Björn Wittenmark. "Sistemas controlados por computador", ISBN:84-283-1593-0, Paraninfo, 1988
- [BUT97] Buttazzo C. "Hard Real-Time Computing Systems". Kluwer academic publishers, 1997
- [BUT02] Buttazzo, G. Lipari, G. Caccamo, M. Abeni, L. "Elastic Scheduling for Flexible Workload Management". IEEE Trans. on Computers, Vol. 51, NO. 3, March 2002
- [CER02] A. Cervin, J. Eker, B. Bernhardsson, K.-E. Årzén "Feedback-Feedforward Scheduling of Control Tasks", *Real-Time Systems*, **23:1**, 2002
- [ISI89] Isidori, A. Nonlinear Control Systems. Spring Verlag, New York, 1989
- [LEI03] Leith, D.J., Shorten, R.N., Leithead, W.E., Mason, O., and Curran P. "Issues in the design of switched linear control system: A benchmark study". *Int. Journal of Adaptive Control and Signal Processing*. 2003; 17:103-108
- [MAR01] Martí, P. Fohler, G. Ramamritham, K. Fuertes, JM. Jitter Compensation for Real-time Control Systems. 22nd IEEE Real-Time Systems Symposium, London, UK, December 2001.
- [MAR02] Martí, P. Fohler, G. Ramamritham, K. Fuertes, JM. Jitter Compensation for Real-time Control Systems.

22nd IEEE Real-Time Systems Symposium, London, UK,  
December 2001.

[NIJ90] Nijmeier, H. “Nonlinear dynamical control systems”, Springer-Verlag, 1990.

[PHI84] Philips, C.L and Troy, N. G.. “Digital control systems, analysis and design”, Prentice-Hall, Englewood Cliffs, N.J., 1984.