PERGAMON

Control Engineering Practice 7 (1999) 445–458

Real-time control of non-uniformly sampled systems

Pedro Albertos[a,]*, Alfons Crespo[b,1]

[a] Department of Systems Engineering and Control, Universidad Politécnica de Valencia, P.O. Box 22012, E-46071, Valencia, Spain
[b] Department of Computer Engineering, Universidad Politécnica de Valencia, P.O. Box 22012, E-46071, Valencia, Spain

Received 12 September 1997; accepted 22 October 1998

Abstract

Industrial applications of digital control require a synergy between well-designed control algorithms and carefully implemented control systems. The design of digital controllers should not be constrained to the simplest case of a single rate with sychronous sampling strategies, and the digital implementation should take account of the actual real-time operating environment. In this paper, control design techniques under non-uniform sampling schemes are reviewed, and computer improvements to cancel or reduce the effect of delays and process interactions are presented. © *1999 Elsevier Science Ltd. All rights reserved.*

Keywords: Time delays; Sampled-data models; Multirate controllers; Real-time systems; Optional tasks

1. Introduction

In classical digital control system design, all the sampled-data signals are assumed to be regularly, synchronously, and equally time spaced, T being the sampling period (Isermann, 1989). However, this is not the case in most industrial applications. There are many reasons to force the use of a different sampling schemes. For instance, the control signal may be updated at a fixed rate while the output vector components are measured by different sensors, each one perhaps having a different sampling rate, noise characteristics and reliability. In some practical cases the output is not available at every sampling time. Another interesting and common situation is in distributed control where multiloop control systems, in which sampling rates are not the same and not even synchronised, are cooperating. This situation also raises the use of multirate control schemes, where the selection of the sampling rate is based on its suitability to control each controlled variable (Berg et al., 1988). This imposes the use of discrete-time (**DT**) models with different sampling periods, even for the same process. Also, for the variety of implemented control activities, such as regulation, adaptation, mode control, supervision, fault detection or optimisation, different time scales and treatments are required.

The use of complicated control algorithms, even when installed in powerful digital systems, takes more time to deliver the control action, introducing time delays in the control loop. These delays, if not accounted for in the design of the control action, may degrade the controlled system performances (Mita, 1985). Also there are controlled systems having just a set of event-driven sensors that trigger an interrupt when the output reaches a certain (known) value or set of values, not necessarily at a sampling instant.

All these sampling patterns may be implemented in systems which are controlled as shown in Fig. 1, where a single-computer-based controller must cope with all the tasks.

The processes to be controlled are continuous-time (CT) systems, and the control subsystem is a discrete-time (DT) one. At the interface, there are a number of samplers, $S_1, S_2, \ldots, S_p$, and hold devices, $H_1, H_2, \ldots, H_m$, working at different instants of time, either periodically, sporadically or magnitude driven, being either independent or coordinated, both in time and in treatment (data), with a common database (DB). Some of these interface activities, mainly those linked to supervision and decision-making procedures, are based on binary signals leading to hybrid dynamic systems. In this paper, which will be focused on regulation and tracking control tasks, a sampled-data model of the process is required.

*Corresponding author. Tel.: + 349-6-387-9570; fax: + 349-6-387-9579; e-mail: pedro@aii.upv.es.

[1]E-mail: alfons@aii.upv.es.

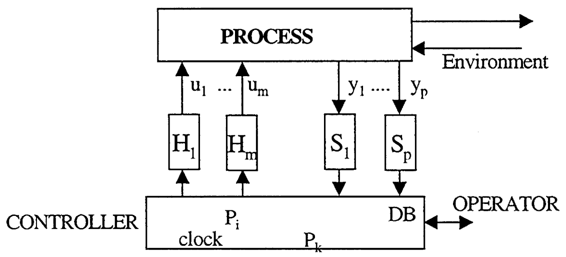


Fig. 1. Complex controlled systems.

From a computational point of view, a number of issues should be taken into account. Other than the numerical problems attached to any digital computation, such as round-off errors, word length limitations, data structuring and explosion, and so on, real-time facilities should be provided. To get the correct answer, on time, requires, at least, knowledge of time attached to any information, the merging of data from different sources, the sharing of data between different processes, a capability for degraded functioning, and interaction with the user in a friendly way to monitor the system behaviour. A number of procedures, P_i, coexist in the same processor, driven by the same *Clock*, and may share information and time constraints. Basic problems in the implementation of digital control should be taken into account:

- Minimising the time delay in generating actions.
- Dealing with distributed data-acquisition (DA) systems.
- Non-synchronous sampling.
- Missing of some samples due to computer overload or emergency operation.
- Sharing data from different resources. Updating of common data among processes, with associated time intervals.
- Data validation under missing data scenarios.
- Interaction and event-driven procedures.

The paper is structured as follows. First, a number of non-uniform sampling schemes are described, and tools to design appropriate controllers are discussed. In particular, the following items will be considered:

(i) variable sampling period,
(ii) non-synchronous sampling,
(iii) multirate sampling,
(iv) missing data, and
(v) event-driven control.

Second, time requirements are considered, and their influence on the control algorithms is evaluated. Some computer software improvements to cancel or reduce the effect of delays and process interactions are discussed. Some final remarks complete the survey.

2. Control issues

As previously mentioned, in a general control scheme the existence of various processes (multiprocess), with many control variables and sensors (multivariable), involving many control loops (multiloop), and operating at different sampling rates (multirate) must be considered. The general problem can be approached in different ways: (1) assume everything is regular, develop control algorithms and apply them at the scheduled time, trying to minimise the delays, (2) assume the actual situation, and develop control algorithms based on the detailed process model, (3) generate software mechanisms to make available the best data obtainable at the time it is required, (4) provide different levels of data: the relevant ones at the maximum precision, and complementary ones at lower precision. Make these levels application-dependent.

Although many of these features may appear together, they are analysed separately below.

2.1. Variable sampling period

A common approach to designing a digital controller is first to design a CT one and then to discretise it. Of course, the coefficients of the DT controller are functions of both the sampling period and the discretisation approach. Moreover, if the sampling interval is not regular, each discrete control action should be evaluated according to the time interval since the last samples. In order to demonstrate the approach, consider two typical controllers, the classical PID controller and the state feedback controller.

PID controller: The CT control action is given by

$$u(t) = K_p e(t) + K_d \frac{\mathrm{d}}{\mathrm{d}t} e(t) + K_i \int_0^t e(\tau)\,\mathrm{d}\tau. \tag{1}$$

A DT counterpart, assuming a sampling period T and a zero-order hold device, is

$$u_k - u_{k-1} = q_0 e_k + q_1 e_{k-1} + q_2 e_{k-2}, \tag{2}$$

where the subindex k stands for time kT, that is, $u_k = u(kT)$. The coefficients q_i, if applying an Euler approximation such as

$$u_k = K_p . e_k + \frac{K_d}{T}(e_k - e_{k-1}) + K_i . T \sum_{j=0}^{k-1} e_j$$

are given by

$$q_0 = K_p + \frac{K_d}{T}; \qquad q_2 = \frac{K_d}{T},$$

$$q_1 = -K_p - 2\frac{K_d}{T} + K_i T, \tag{3}$$

showing their dependence on the sampling period. Simple formulas derived from Eq. (3) allow the controller coefficients to be directly updated if the sampling period changes, and similar expressions can be derived for any DT transfer function (Salt and Albertos, 1990). If the sampling is irregular, taking place at the time sequence $\{\dots, t_{k-2}, t_{k-1}, t_k, \dots,\}$, these coefficients are given by

$$q_0 = K_p + \frac{K_d}{t_k - t_{k-1}}, \qquad q_2 = \frac{K_d}{t_{k-1} - t_{k-2}},$$
$$q_1 = -K_p - \frac{K_d(t_k - t_{k-2})}{(t_k - t_{k-1})(t_{k-1} - t_{k-2})} + K_i(t_k - t_{k-1}). \tag{4}$$

Similar expressions are obtained if any other discretisation approach is used to get Eq. (3) from Eq. (1).

State feedback control: Assume a conventional state-space process model:

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{5}$$

and a control law $u(t) = K.x(t) + r(t)$, where $r(t)$ is the external input. If a regular sampling period T is applied, to retain equivalent closed-loop poles (only valid for small time intervals), the DT feedback law should be (Aström and Wittenmark, 1984)

$$u_k = K'.x_k + r_k = (K + K^*T).x_k + r_k \tag{6}$$

where $K^* \cong K.(A_c + B_c K)/2$.

If the sampling is irregular, the same DT control law can be applied, taking $T = t_k - t_{k-1}$.

2.2. Delayed updating/sampling

Basic CT control design approaches rarely take account of time delays. Time delays can appear in the process and/or in the actuators, and sometimes in the sensor instrumentation when the measurement requires some sort of transportation or handling. In any case, these delays are usually considered as part of the process to be controlled. This is not the case when digital controllers are used. Besides the above-mentioned delays, some sources of unavoidable time delay appear: analogue-to-digital conversion, control algorithm computation, D/A conversion and control-signal interfacing as well as the updating of digital actuators. If the required performance of the controlled process is very stringent or the processes to be controlled are very fast, the sampling period must be shortened, and these delays may degrade the system control performance if they are not considered in the design stage.

A similar situation happens in the control of open-loop unstable systems. Academic examples, like the inverted pendulum, or the control of under-damped mechanical structures under vibrations, pose the same kind of problems: delays in the control action may result in system instability. It is possible to show that, if a great control effort has been used to get the nominal control, very short-time delays become significant, as the system is operating in an open loop. An interesting work (Mita, 1985), in the framework of optimal control, assumes a one-sampling-period time delay to design and implement the proper controller, making the system more sensitive than necessary to plant parameter uncertainties and external disturbances.

To simplify the mathematical treatment, it is assumed that all the measurements are obtained simultaneously but, to allow their processing time, they are acquired Δ seconds before the control action is applied, as shown in Fig. 2.

Consider the effects of these time delays as they would occur in CT systems, like Eq. (5). If a linear state feedback control law is designed, the nominal controlled system will be

$$\dot{x}(t) = (A_c + B_c K)x(t) + B_c r(t). \tag{7}$$

Assume a delay of Δ units of time in the generation of the control action. The complete system will be

$$\dot{x}(t) = A_c x(t) + B_c K x(t - \Delta) + B_c r(t - \Delta). \tag{8}$$

The open-loop transfer function matrix will be

$$G(s) = K(sI - A_c)^{-1} B_c . e^{-s\Delta}. \tag{9}$$

For single-input/single-output (SISO) systems, if K were obtained from an optimal LQR design, a phase margin P_M of at least $\pi/3$ rad is assured for the nominal design (Kalman, 1964). This would mean that, to keep the system stable, the maximum undesired time delay is $\Delta_m = P_M/\omega_f$, where ω_f is the maximum frequency such as $|G(\omega_f)| = 1$. Of course, this delay would be unacceptable because a minimum phase margin is always required.

Now, assume that this control law is digitally implemented. As seen in Eq. (6), the feedback matrix should be redesigned to take account of the sampling time T, and the system performances are maintained if the sampling period is short enough. Under these assumptions, Δ_m establishes an upper limit on the sampling time. This extreme situation warns about any delay that would be significant with respect to the sampling period, if it has been chosen to be very tight.

For $\Delta \le T$, the DT model for the sampled-data system in Eq. (8) can be expressed by an augmented state vector

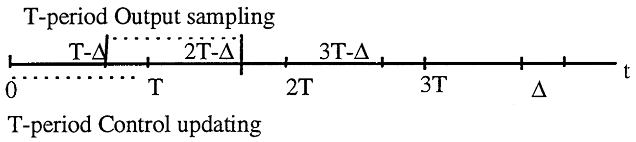


Fig. 2. Delayed updating/sampling.

including the delayed input:

$$\begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} A(T) + B(T-\Delta)K & A(T-\Delta)B(\Delta) \\ K & 0 \end{bmatrix} \cdot \times \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} B(T-\Delta) \\ I \end{bmatrix} r_k, \tag{10}$$

where

$$A(T) = e^{-A_c T}, \text{ and } B(T) = \int_0^T e^{-A_c \tau}\,\mathrm{d}\tau . B_c.$$

Of course Eq. (10) can be generalised for any greater value of Δ, with a larger state vector. Two problems may be considered: (a) to study the effect of Δ on the stability of Eq. (10), assuming that the undelayed system is stable, (b) given a desired closed-loop behaviour defined over the initial CT system, to investigate how the control structure can counteract the presence of this delay to get the performances that are as close as possible to those of the undelayed system. The first problem is analytically infeasible in the general case. Some general ideas can be obtained for the scalar case. In multivariable systems only very particular situations can be solved. On the other hand, some computer-aided design tools, such as those presented in Salt et al. (1993), allow an easy simulation analysis of this problem. To solve problem (b), an advanced observer or predictor is defined, based on the intrinsic properties of sampled-data systems. The proposed solution can be applied even with variable time delay.

Advanced observer (*predictor*): As shown in Fig. 2, it is assumed that the control action is applied in nominal time but, to prevent digital delays, the measurements have been taken Δ seconds earlier. Assume, first, that the complete state is measurable. Denote by $x_{x,\Delta}$ the state measured at time $kT - \Delta$. The state at kT could be reconstructed by

$$\bar{x}_k = A(\Delta)x_{k,\Delta} + B(\Delta)u_{k-1}. \tag{11}$$

So, the control law will be

$$u_k = K[A(\Delta)x_{k,\Delta} + B(\Delta)u_{k-1}] + r_k. \tag{12}$$

There is no problem if $x_k \cong \bar{x}_k$, but some differences will appear if either there are some uncertainties in the process model (A, B), or there exist some unmeasured disturbances.

A similar result can be stated if the state is not fully measured. Then, assuming state observability, an observer can be built. Denote by $y_{k,\Delta} = Cx_{k,\Delta}$, the output measured at time $kT - \Delta$. Taking into account Eq. (11), a new computed measurement of the state at kT can be expressed by the new output equation

$$z_k = C.A(-\Delta)x_k = y_{k,\Delta} + C.A(-\Delta).B(\Delta)u_{k-1}. \tag{13}$$

The advanced observer will be defined by

$$\hat{x}_{k+1} = A(T)\hat{x}_k + B(T)u_k + K_o(z_k - \hat{z}_k), \tag{14}$$

where K_o is the observer gain, and, similar to Eq. (13), $\hat{z}_k$ is computed by

$$\hat{z}_k = CA(-\Delta)\hat{x}_k. \tag{15}$$

It is standard to prove that if the pair $[A(T), C]$ is observable, the pair $[A(T), CA(-\Delta)]$ is also observable, (Chen, 1984). The Luenberger-type observer scheme is that of Fig. 3.

Following the classical observer design techniques, to avoid the use of delayed information, the observer may be slightly changed in such a way that the output includes the most actual measurement. In a similar way a reduced-order observer can be developed.

It is important to notice that observers built up without the current information can greatly degrade the control performances. It must be pointed out that in this approach no reference is made to how K, the controller gain, or K_o, the observer gain, are obtained. Any useful methodology, such as pole placement, dead beat, LQG optimal design or some other design resulting in a linear control law, can be applied. The observer in Eqs. (14)–(16) will reduce the problem of delays in digital controllers.

2.3. Multirate sampling

It is well known that in digital control, the faster sampling rate is not always the best one, due to numerical and resolution problems. Thus, the selection of the sampling rate is a trade-off between loss of information and the processing capability of the digital system. As previously mentioned, in many control applications, due to the time scale of the subprocess dynamics, or some other technical reasons, such as hardware constraints, the

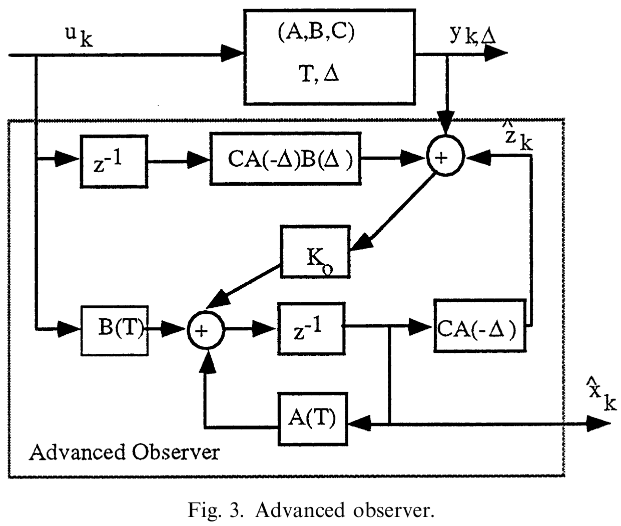


Fig. 3. Advanced observer.

sampling rate is not the same for all the process variables. That implies the use of different DT models even for the same CT process. Moreover, in process industry applications, the control updating rate can be faster than the measurement output rate, due to measurement system constraints, such as discontinuous analysers and so on.

To deal with this situation in a simplified way, it is assumed that, although the sampling rate for all the variables is not the same, the whole system is periodic, denoting by T_o the overall interval periodicity. The sampling scheme is drawn in Fig. 4, where, in the framework time period T_o there are s output measurements (the last one at $(k+1)T_o$), and r input updates (the first one at kT_o). In general, if the system is DT controlled, it is T-period open-loop and step-input controlled. If during a control period more information can be obtained, it can be computed more efficiently. If the control action can be more frequently updated, u_k^i, better CT response would be achieved.

There are a number of techniques to represent such kinds of systems, mainly based on enlarging the number of components of the input and/or output vector and transforming the model into a single-rate one. The lifting approach (Voulgaris and Bamieh, 1993) or the block multirate input–output (BMIO) model (Albertos, 1990) are quite similar.

Following Albertos (Albertos, 1990), consider a n-order SISO, with $T_o = NT$, and periodic, regular and synchronous sampling. If the T-shift operator is denoted by q, the T-period difference equation for Eq. (5) is

$$A(q^{-1})y(lT) = B(q^{-1})u(lT), \tag{16}$$

where

$$A(q^{-1}) = 1 + \sum_{i=1}^{n} a_i q^{-1}; B(q^{-1}) = \sum_{i=1}^{n} b_i q^{-1}, \quad b_1 \neq 0$$

and the DT state-space equation is

$$x[(l+1)T] = Ax(lT) + bu(lT); \qquad y(lt) = cx(lt), \tag{17}$$

where $A = A(T)$, $b = B(T)$.

By forming the one-sampling-period-shift vector blocks:

$$Y_k = \begin{bmatrix} y(kT_o + T) \\ \vdots \\ y(kT_o + NT) \end{bmatrix}; \qquad U_k = \begin{bmatrix} u(kT_o) \\ \vdots \\ y(kT_o + NT - T) \end{bmatrix}, \tag{18}$$

where k stands for the kT_o-period, the following recursive output computation is obtained:

$$Y_k = \bar{A}Y_{k-1} + \bar{B}U_k + \alpha Y_{k-1} + \beta U_{k-1}, \tag{19a}$$

where all matrix entries depend on a_i, b_i, the transfer function polynomial coefficients. Alternatively, it can

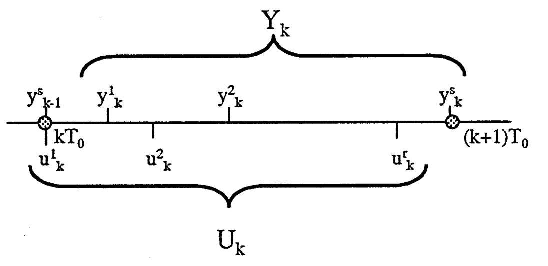


Fig. 4. Multirate sampling.

also be expressed by

$$Y_k = PY_{k-1} + QU_{k-1} + Q_1 U_k. \tag{19b}$$

Similarly, the N-period discrete state-space representation is

$$x_{k+1} = x[(k+1)T_o] = A^N x_k + WU_k, \tag{20}$$

where $W = \lfloor A^{N-1}b \dots Ab \quad b \rfloor$, and the output block is given by

$$Y_k = O.x_k + H.U_k, \tag{21}$$

where

$$O = [A^{\mathrm{T}}c \cdots A^{N^{\mathrm{T}}}c]^{\mathrm{T}} \quad H = \begin{bmatrix} h_1 & & 0 \\ \cdots & \cdots & \cdots \\ h_n & \cdots & h_1 \end{bmatrix},$$

$h_i = cA^{i-1}b.$

Thus, Eq. (19) is equivalent to the following expression:

$$Y_k = OA^N O^{\#}.\ Y_{k-1} + O[W - A^N O^{\#} H].U_{k-1} + H.U_k, \tag{22}$$

where $\#$ denotes the matrix pseudoinverse. From Eq. (20), assuming a desired closed-loop response such as $x_{k+1} = A_d x_k$, the regulation problem is solved by applying the control law:

$$U_k = W^{-1}[A_d - A^N x_k], \tag{23}$$

and the tracking problem is solved, for the reference Y_d, by the alternate control

$$U_k = Q_1^{-1}[Y_{d,k} - PY_{k-1} - QU_{k-1}]. \tag{24}$$

In this case, if the same period is used for both the measurement sampling and the control updating, a receding horizon control can be foreseen. The first control action in Eq. (18) is applied and the control block (24) is computed for any T. A more general BMIO model for any periodic sampling pattern may be derived, and used in Eq. (24), (see Albertos, 1991) for details).

Dual-rate control: An interesting particular case is that of different measurement/control updating rate. In a recent paper, (Albertos et al., 1996), a process model to deal with the maximum available information from the process has been presented. The dual-rate operator,

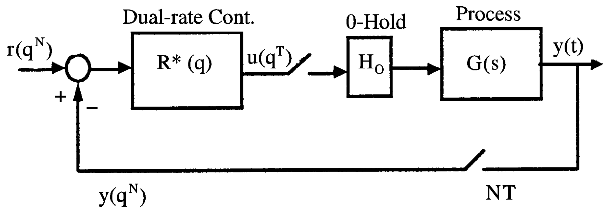


Fig. 5. Dual-rate system.

developed for SISO systems, is briefly introduced. Faster control updating than output measurement, as shown in Fig. 5, is considered.

Denoting by q^N the NT-shift operator, the following operators can be defined:

$$\text{Fast rate:}\quad G_1(q)=\frac{y(q)}{u(q)}=\frac{N_1(q)}{D_1(q)}. \tag{25}$$

$$\text{Slow rate:}\quad G_N(q)=\frac{y(q^N)}{u(q^N)}=\frac{N_N(q^N)}{D_N(q^N)}. \tag{26}$$

$$\text{Dual-Rate:}\quad \bar{G}(q,N)=\frac{y(q^N)}{u(q)}=\frac{\bar{N}(q,N)}{D_N(q^N)}. \tag{27}$$

Denoting by $W(q)=D_N(q^N)/D_1(q)$ the denominators ratio, the following equalities hold:

(1) $\bar{N}(q,N)=N_1(q).W(q)$.
(2) $\alpha_{i,N}=\alpha_i^N$, slow and fast models' poles.
(3) Coefficients of $N_N(q^N)$ are partial sums of those of $\bar{N}(q,N)$.

A dual-rate control may be computed as

$$R^*(q)=\frac{1}{1-M(q^N)}\cdot\frac{1-q^{-N}}{1-q^{-1}}\cdot\frac{M(q)}{G(q)}, \tag{28}$$

where $M(.)$ represents the desired output/input controlled transfer function. The control scheme is composed of a slow control component, a sample-and-hold device to convert the rates, and a fast control element (Fig. 5).

This model also allows for a dual-rate adaptive control. The options are shown in Fig. 6. By a standard parameter-estimation algorithm (LS in the figure), the dual-rate model parameters are estimated. The above model parameter equalities allow the computation of the single-rate model parameters, either the fast or the slow rate one, which are used for the controller tuning and/or design.

2.4. Missing data

As previously mentioned, in some industrial applications the output is not available at every planned sampling instant due to computer overload, communication errors, shared sensors or event-driven sensors. The simplest solution is just to take the last gathered data, but this could lead to erroneous control computation. This

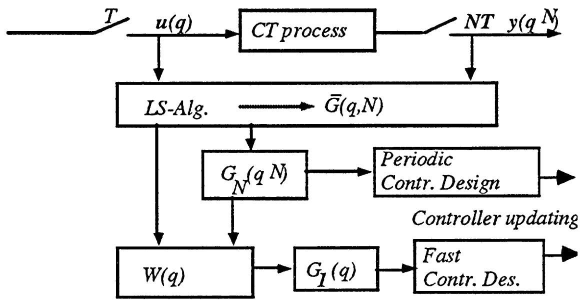


Fig. 6. Dual-rate adaptive control: identification and control updating.

problem can be also treated by using a process model running in parallel. This model should be fed with all the available measurements, and can be used to provide updated information, to improve the process model, or as a basis on which to generate the control actions. Modifications on the basic least-squares algorithms used in conventional sampled-data control and fault detection, (Isermann, 1989), have to be carried out in situations of missing data (Albertos et al., 1997b).

The proposal here is to update the current model estimate by applying an RLS algorithm. If no measurement is available for an output component of the regression vector, the output of the current estimated model replaces it. Under some assumptions (Albertos et al., 1997b), unbiased convergence can be proved. Again, for the sake of simplicity, a SISO system is assumed. Output equation (16) can be rewritten as

$$y_k^*=\psi_k^T\theta, \tag{29}$$

where θ is a parameter vector

$$\theta=[a_1\quad a_2\ldots a_n\quad b_t\ldots b_n]^T \tag{30}$$

and ψ_k is the regression vector

$$\psi_k=[-y_{k-1}\cdots-y_{k-n}\quad u_{k-1}\ldots u_{k-n}]^T. \tag{31}$$

The actual measurement is $y_k=\psi_k^T\theta+\eta_k$, where η_k is a white noise sequence. Assume that the available data reflect past control actions and the plant output at a set of instants $\{k_1T, k_2T, \ldots\}$. If all k_j are integers, then a synchronous sampling situation is present. If the sampling is asynchronous, k_i will be real. The use of a state observer to update the model state on the basis of the current measurement may improve its performances. The modified missing-data RLS estimation algorithm (Albertos et al., 1997b), is stated as follows (synchronous case):

1. Update the model (29) with the current parameter estimates.
2. Apply to the model (29), the same input signal than that of the process.
3. If there is no sample available at this sampling instant assume the output from the system is the output from the model, and go back to step 2; else take the process measurement and go to step 4.

4. Form regression vectors (31) with past inputs and outputs (from either the process or the model). Calculate the prediction error of the model with respect to the process measurement. Carry out one LS algorithm iteration and go to step 1.

Inserting actual measurements in regression vectors and calculating future model outputs is the most straightforward approach, but may yield an unstable observer scheme (Albertos et al., 1997b).

2.5. Event-driven measurements

The previous algorithm has to be modified when dealing with random sampling of continuous processes. It may be assumed that no measurement happens at exactly the instant at which the input signal is being updated, so the previous procedure is not applicable. Now the steps are modified in the following way:

3a. If there is no sample till next sampling instant, go to step 2.
4a. Assuming an intersampling measurement is present, produce an intersampling response prediction from the model and calculate the prediction error. Apportion this error to the immediately previous and immediately subsequent sampling instants (see later for details).
5a. Carry out an LS identification algorithm with the input–output pair from the previous iteration (to ensure that no further prediction error is apportioned to it in the ensuing intersampling period). Use that previous iteration information as an input to a suitable state observer. Update the model with the current parameter estimates. Go to step 1.

So the RLS algorithm will be executed as before but assuming that two fictitious sensor measurements were available. To predict the intersampling response in step 4a several methods can be used:

- *Constant* extrapolation: take the output from the last fictitious sampling instant.
- *Linear* interpolation: predict the output at the next sampling instant, and get the prediction on the straight line that joins those predictions.
- *Parabolic* and higher-degree interpolation.
- *State-space* extrapolation, with fractional powers of DT state-representation matrices.

In the application of the algorithm, linear interpolation may be good enough.

3. Real-time issues

Control applications require that several parallel activities be defined to model the environment. Periodic tasks model the activities that must be executed at periodic instants of time. Sporadic tasks model the activities that handle significant external events. While the process of control design is focused on obtaining the regulator, later translated into an algorithm, the software design is focused on producing pieces of software that will be executed concurrently under the supervision of a scheduler.

3.1. Task model

A standard control loop implementation written in Ada95 has the scheme shown in Fig. 7.

The code is composed of two parts: specification and implementation. In the specification part the parameters associated with the task (initial_time, period and phase) are defined, while in the implementation part an infinite loop that executes at each period the actions related to the control (get external values, calculate the action value, send it, and compute global state) is included.

This code corresponds to a classical control scheme where the computation of the control action can be undertaken using traditional techniques. However, more and more applications require complex computation and the use of exhaustive algorithms (unbounded algorithms) that can compromise the response time of the system. If this is the case, the computation time should be split into a fixed (mandatory) part and an optional part (improvement).

```
task Level_Control (initial_time, period, phase: TIME);

task body Level_Control is
level : Sensor_Value;
action : Actuator_Value;
reg : Regulator;
next_period : TIME;        -- period  task attribute

begin
Define_Regulator(reg, par1, par2, ...);
....
delay until (initial_time + phase);
next_period := initial_time + phase;
loop
   level := get_level_sensor();
   Regulator_evaluate(reg, level, action);
   -- operations to improve the regulator results
   send_actuator(action);
   -- operations to evaluate the global state
   --operations to prepare the data for the next sampling
   -- operations of updating the global data base
   delay until next_period;
   next_period := next_period + period;
end loop;
...
end Level_Task;
```

Fig. 7. Standard control-loop implementation in Ada95.

Assuming that the acquisition of external information is usually done by external devices, without requiring CPU time, the activities to be done by the task code can be structured in the following parts:

1. Computation of the control action. This is mandatory, and should be executed as soon as possible. At the end of this part, the basic control action is ready to be sent to the actuators.
2. Solution improvement. During this time, an algorithm or method is able to improve the answer obtained in the first part. This computation may be based on an iterative technique or on other methods (knowledge based methods, progressive reasoning, etc.). At the end of, or during, this second part, an improved solution to be sent to the actuators is available. This part can be considered as optional.
3. Output the control action. This must be done either as soon as possible, or within a fixed time interval.
4. Evaluation of the global state and preparation of the data for the next period. This part is required to be executed before the next period, but at any time within the period.
5. Updating the global information (data base, graphical displays, etc.).

These activities can be represented in a graphical way, Fig. 8, where δ represents the DA system delay and Δ the control/measurement one.

Activities 1 and 3 must be executed before the deadline of the task. Activity 2 is optional, and can be executed as far as there is available time. Activity 4 must be executed before the next sampling period. Activity 5, involving a number of operations such as data-base updating or operator communications, can be delayed until some free time is available. It is a maintenance activity that can be done at any time. These activities are executed by a centralised service, and they will not be considered further.

In this case, the number of tasks is known in advance and their scheduling may be analysed. The main concerns being to fix and keep as short as possible the time used in activities 1–3, and the timing between tasks, if these must share data and/or results.

Other than the response time, some timing constraints associated with real-time tasks should be considered. In this way, attached to a task T_i $[T_i = (C_i, P_i, D_i, \mathrm{Pr}_i, \Phi_i)]$ the following parameters are defined: the worst case execution time (WCET), C_i, the period, P_i, the deadline, D_i, the assigned priority, Pr_i, and the phase or offset, Φ_i, which is the delay of the task period with respect to the overall period. The global execution time for this model is represented in Fig. 9.

In a real-time system, the designer can define several tasks associated with different control loops. If a task is divided into N parts, C_i will be defined as the sum of the times required for all task parts ($\sum_{j=1}^{N} C_i^j$). In the case of mandatory and final parts, it corresponds to the WCET.

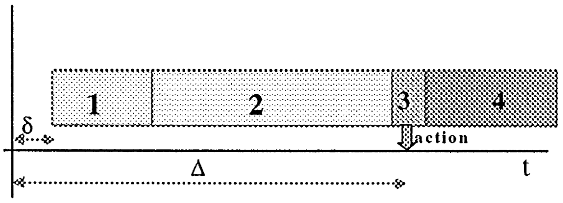


Fig. 8. Delay in actions.

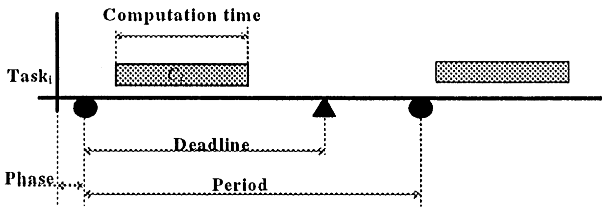


Fig. 9. Task timing.

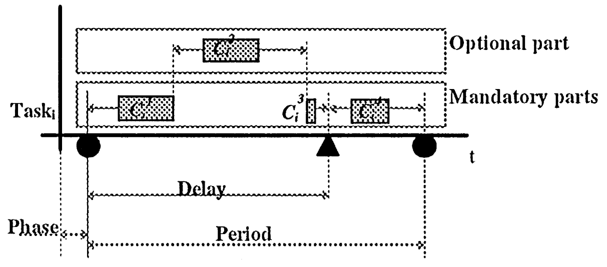


Fig. 10. Task block timing.

In the case of optional parts it is assumed as the average execution time, which is calculated on-line from the input parameters of a specific invocation, or it is taken as the average of previous time executions. The deadline specifies the maximum time to complete the system reaction. If the control action, the reaction of the system, is not produced at the end of the computation time, the deadline can be interpreted as the time at which to send the control action to the actuator instead of for the complete computation. With the task structure defined above, the timing constraints can be depicted as in Fig. 10.

3.2. One-shoot sampling scheme

Consider a control system with three single loops. Assuming the basic model (Fig. 9) and the same period for all the tasks, fixed or variable, the easiest solution is to define a cyclic scheduler, where these tasks are sequentially executed, as shown in Fig. 11. To determine the task schedulability, the constraint $\sum_i C_i < T$ must hold. As the task execution order is always the same, the control updating/measurement sampling delay may be known in advance and, if required, taken into account in the control algorithms (like Eq. (12)).

```
with COMPONENTS_CONTROL ;
use COMPONENTS_CONTROL;

PERIOD : TIME ;
T : TIME := CLOCK ;

procedure Temp_Control is
temp: Sensor_value; pvt : Actuator_Value;
S1: Sensor; A1: Actuator; reg1: Regulator;
begin
   temp := Get_Sensor(S1);
   Determine_Control_Action(t, pvt, reg1) ;
   Send_Actuator(pvt, A1) ;
end Temp_Control ;

procedure pH_Control is ...

procedure Level_Control is ...

begin
 -- variable initialisation
 loop
                Temp_Control;
                pH_Control ;
                Level_Control;
                PERIOD := PERIOD + T ;
                delay until PERIOD ;
 end loop;
end Control_System ;
```

Fig. 11. Cyclic scheduling of tasks.

With this simple scheme three sequential activities (procedures) are executed each period. Thus, a cyclic scheduler has been defined in Fig. 12.

If the computation time of one task is significantly shifting the execution of the following tasks, a phase may be attached to them and the DA command may also be delayed in the same interval. These would result in a delayed sampling scheme, as already discussed.

This solution is not very convenient because a high number of situations can be produced. These could include: an error in a task can stop all the control loops, sporadic handling of events or interrupts might not be allowed, and many others.

If the sampling period is not constant, but the same for any task, the only difference from the previous case is the control algorithm, requiring to record the delay involved in the tasks, Δ, as this will be used in task 1 (see algorithm (4)). The control design should ensure, in the worst case, a minimum sampling period that is longer than the time required to carry out these activities.

3.3. Multirate sampling

If there is a different period for each task, or for groups of them, a cyclic scheduler is still possible, but the solution is more complex and less flexible. The number of possibilities to distribute all the tasks into a hyperperiod is very high, and the timing analysis and task scheduling should be decided a priori.

In this case it seems more attractive to use a concurrent environment, and to define an appropriate policy to

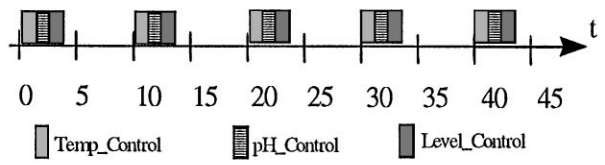


Fig. 12. Chronogram for this example.

execute all the tasks. In real-time systems the scheduling algorithm has to guarantee the timing constraints of the tasks. The designer can define several tasks, associated with different control loops.

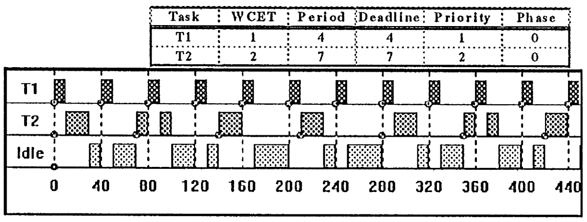
Most of the issues pointed out in the introduction may and should be covered by the control algorithm, as discussed in the previous section. Two crucial issues related to the measurement/action delay and the performing of complementary tasks are strongly influenced by the scheduler design and will be analysed in the following section.

4. Time constraints and optimisation

In recent years, there has been great interest in using fixed-priority pre-emptive scheduling as a technique to determine whether the timing constraints will be accomplished or not (Liu and Layland, 1973; Audsley, 1991; Burns, 1993). One of the advantages of this engineering practice is the system's ability to determine whether it is schedulable or not, based on a few parameters involving periods, deadlines, and WCET. The periods are determined by and depend on the dynamic of the system, while the WCET is the measure of the time needed for the execution of some required actions. In practice, this parameter can be experimentally measured by isolating the activity. However, even if the system is schedulable, some action delays can modify the system behaviour.

4.1. Delays

Once these parameters have been determined for each process, it is convenient to analyse the effects of the multitasking on the control action delay. The execution chronogram of a system running two tasks, with the characteristics as described in the attached table, is shown in Fig. 13. The multitasking effects on data acquisition and output have been studied and considered in the scheduling. In addition to the multitasking effects, other aspects such as delay in the data acquisition, transmission lines, etc., define the jitter on the input or output. Several schemes can be found in the literature to reduce the effects of the jitter (Klein et al., 1993; Gonzalez et al., 1991). In this part, an analysis of the effects of multitasking on the output will be detailed, using the task model



Task	WCET	Period	Deadline	Priority	Phase
T1	1	4	4	1	0
T2	2	7	7	2	0

Fig. 13. Two-task chronogram.

described above. These tasks have a single mandatory part, and the action is done at the end of the computation ($Ci = C_i^1 + C_i^3$).

As a result of this scheduling, the higher-priority task (T1) is selected by the operating system as soon as it is ready to be executed. In this case, if it is considered that the action is delivered at the end of the computing time, it will be always sent with the same delay. Of course, a small interval should be allowed, due to the consideration of variations in the execution of the task. However, the interval of T2 depends on its simultaneity with the execution of T1. In the worst case, the action provided by T2 is delayed during the complete execution of T1, whereas the best situation occurs if no simultaneity is present.

The effects of the delay on the execution of both tasks is shown in Fig. 14. The starting and finishing task intervals probability for T1 and T2, with respect to their own periods, are shown in the graphic on the left. T1 always (100%) starts and finishes (1/4 of its period) on time. On the other hand, T2 starts its execution at the beginning of its period only 75% of the time, while for 25% of the instantiations it starts after a delay of 1/7 of its period (14,28% or 10 time units, TU). T2 finishes either at 20 or 30 TU (28 or 43% of its period). Therefore, the control action produced by T2 is delivered within a time interval of 10 time units. This time interval corresponds to the jitter on output and will be denoted as the *control action interval* (CAI). In the graphics on the right, the effects of a 20% variation, with respect to the WCET, of the time execution of all tasks is considered. As a conclusion, the T1 task will finish within a CAI of 5% [20%, 25%], whereas the task T2 has two CAIs of $\pm$ 2.5% around 28 and 40% of its period.

This concept may be extended to any control action. Ideally, the delay should be constant, to properly apply algorithms like those obtained by the advanced predictor (Section 3.2), but some time margin should be allocated for this action. The CAI can be analytically determined from the WCET of the tasks. In the above example, the $\mathrm{CAI}_{\mathrm{T1}}$ is either 0% when T1 consumes all the WCET or 5% of its period when there is a 20% variation in its execution time with respect to the WCET. The $\mathrm{CAI}_{\mathrm{T2}}$ is either 14 or 17% of its period.

If the number of tasks increases, the CAI of the low-level priority tasks is also increased. The chronogram and the CAI values for a set of four tasks, with the parameters as defined in Table 1, are shown in Fig. 15.

The effect of the delays in the task set (T3 and T4), as described in Fig. 15, is shown in Fig. 16. For instance, T3 has a WCET of 35 TU (23% of its period) but it never

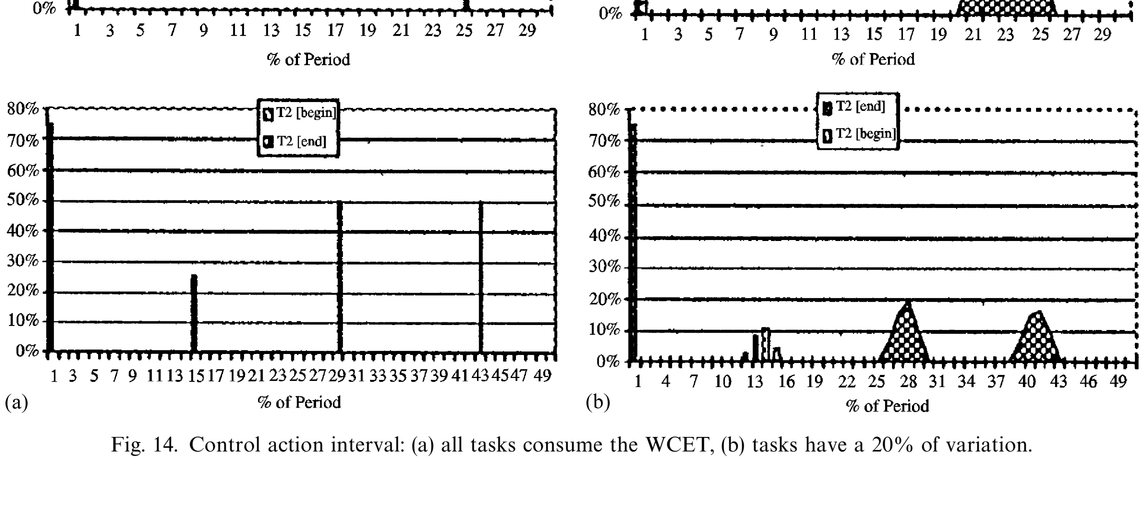


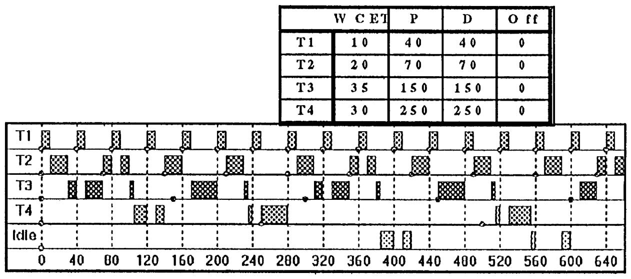
Fig. 14. Control action interval: (a) all tasks consume the WCET, (b) tasks have a 20% of variation.

Table 1
Interval variation of beginning and ending times: fixed WCET and 20% variation in the WCET

Fixed	Begin		End		
WCE	Min	Max	Min	Max	CAI %
T1	0	0	10	10	0
T2	0	10	20	30	14
T3	0	30	45	105	40
T4	0	105	30	240	84

20%	Begin		End		
WCET	Min	Max	Min	Max	CAI %
T1	0	0	8	10	5
T2	0	10	18	30	17
T3	0	30	42	101	39
T4	0	101	28	140	45

sends the control action before 45 TU (31%). This corresponds to the best case (faster reaction time for T3). The worst case (slower reaction time) occurs when the task sends the output after 105 TU (70% of its period). The CAI is the interval between the faster and slower response times (39%). Even worse results are obtained for T4 (the less prioritised). It has a CAI of 84% of its period.



	W CET	P	D	Off
T1	10	40	40	0
T2	20	70	70	0
T3	35	150	150	0
T4	30	250	250	0

Fig. 15. Chronogram of the four-task example.

These values are increased and decreased for higher and lower tasks, respectively, if some randomness (20% with respect to the WCET) in the execution time is assumed.

In the preceding sections, it has been considered that only mandatory parts ($C_i^1 + C_i^3$) were defined. Similar results can be obtained if it is assumed that the part specified in the above examples is the sum of all mandatory parts of the activity ($C_i^1 + C_i^3 + C_i^4$).

For instance, in Table 2, the computation time of these parts is included.

As the action is obtained at the end of the first part, the result is sent before completion of the task. In terms of the CAI, it is improved with respect to the above results, but there is still such an interval.

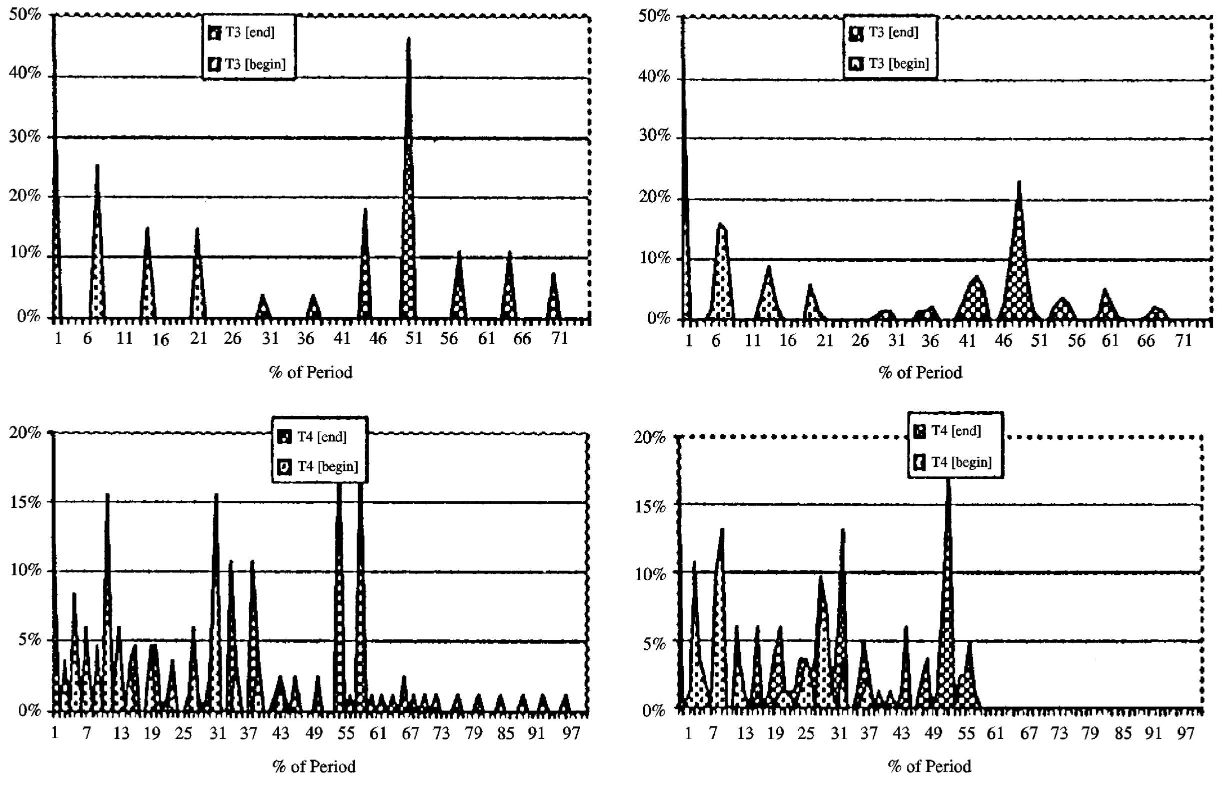


Fig. 16. Four task: timing and delays.

Table 2
Four task parameters

Task	C^1	$C^{3,4}$	Period	Deadline	Priority	Phase
T1	0.8	0.2	4	4	1	0
T2	1.5	0.5	7	7	2	0
T3	3.3	0.7	15	15	3	0
T4	2.5	1.5	25	25	4	0

4.2. Reduction of the CAIs range

In the above examples, the priority assignation is produced by taking the period of the task into account. A more frequent task has assigned a higher priority than a less frequent one. Because the CAI depends on the interference level with other tasks, if a task increases or decreases its priority level, the CAI will consequently be reduced or increased.

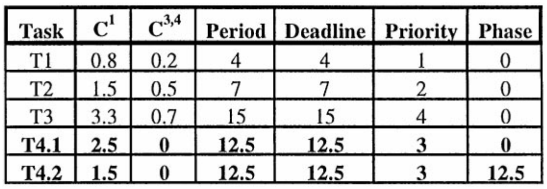
However, in order to apply the basic rate monotonic (RM) or deadline monotonic (DM) theory (Liu and Layland, 1973), the priority of a task should not be increased or decreased, as it is a consequence of its rate or deadline, respectively.

To design the system, several options can be used by the engineer. A first solution is to decrease the deadline of a task maintaining the period. When deadlines are shorter than periods, the deadline monotonic priority assignation can assign a higher priority to the modified task. Using this approach, the system is stressed and the modification can produce a non-schedulable system.

A second approach consists of modifying the priority of a task, and then computing the worst-case completion time of each task to determine whether the system remains schedulable or not (Lehoczky et al., 1989). This solution is not recommended because it is not possible to apply all the properties of the RMA, although it is possible to determine the release time of the tasks to determine the schedulability.

Another approach that does not stress the system consists of a modification of the period, taking advantage of the task scheme presented above. Thus, the changing of the task's priority should be a consequence of a change in the period. From the control point of view, the period cannot be changed, but from the software design viewpoint, some techniques to change the period of a task can be applied. This change will result in a priority change in the system.

To increase the priority of a task, it could be split into two parts, dividing the period by two, and alternately executing one part in each period. In the model previously described, decomposition criterion could be based on the task parts, as shown in Fig. 17. For instance, in the example above, the task T4 could be split into T4.1 and T4.2. According to the task model (Fig. 7), T4.1 may include the first and third parts, and T4.2 the fourth one.



Task	C^1	$C^{3,4}$	Period	Deadline	Priority	Phase
T1	0.8	0.2	4	4	1	0
T2	1.5	0.5	7	7	2	0
T3	3.3	0.7	15	15	4	0
T4.1	**2.5**	**0**	**12.5**	**12.5**	**3**	**0**
T4.2	**1.5**	**0**	**12.5**	**12.5**	**3**	**12.5**

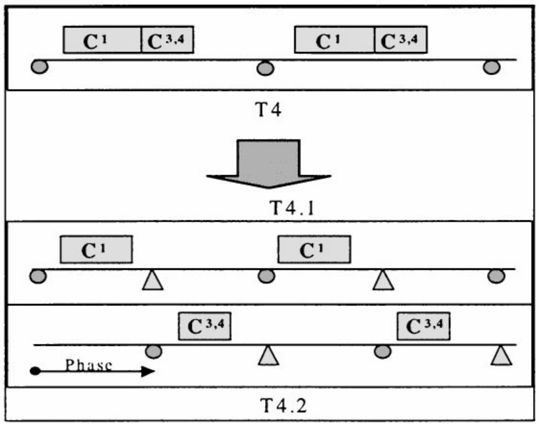


Fig. 17. Task splitting.

At each period, one of the tasks will be, alternately, executed.

The rate of task T4 is higher than that of T3 and, consequently, if the rate monotonic is applied, its priority is adjusted with respect to other tasks. It allows the CAI of T4 to be reduced but, as a result, the CAI of T3 is increased. The schedulability analysis is applicable if the maximum of both parts is considered as WCET. The results will be coherent if the computation times for both parts are similar.

An alternative approach is to increase the period of a task which is less important from the user's point of view, but more frequent than others that are more important. In the above example, to decrease the priority of T3, two copies of T3 (T3′ and T3″) are used over a double period, one of them with a phase equal to the period, Fig. 18.

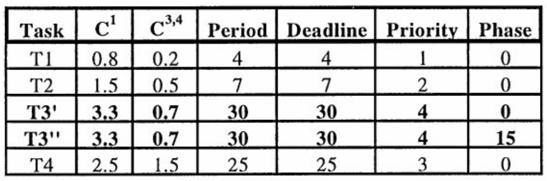
In any case, the reduction of the CAI of one task produces an increment in the CAI of another one.

The first proposed solution is more appropriate for this kind of situation because the activities of the task are decomposed at the design stage, and can be considered as two sequential activities.

4.3. Tasks with optional parts

The inclusion of optional parts in control-system algorithms allows the use of the remaining CPU time to improve the response to some control demands. The remaining CPU time is higher in practice than the theoretically calculated value, because:

- Some sporadic activities are transformed into periodic ones when they have hard real-time constraints, and they are only executed in few periods.



Task	C^1	$C^{3,4}$	Period	Deadline	Priority	Phase
T1	0.8	0.2	4	4	1	0
T2	1.5	0.5	7	7	2	0
T3'	**3.3**	**0.7**	**30**	**30**	**4**	**0**
T3''	**3.3**	**0.7**	**30**	**30**	**4**	**15**
T4	2.5	1.5	25	25	3	0

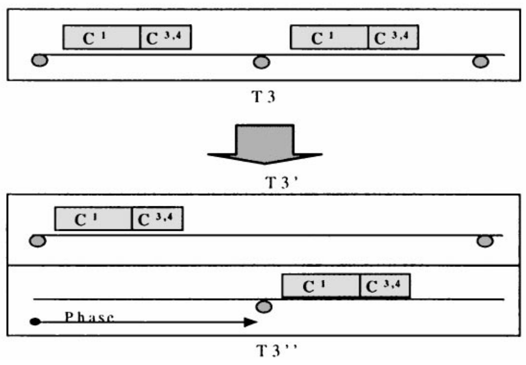


Fig. 18. Task period doubling.

- The computation time considered for each task is the worst case. In most of the periods the real computation time will be shorter.
- When tasks share resources, the WCET considers the maximum blocking time.

All these factors produce a saving in the CPU time that can be invested in the improvement of some results. On the other hand, *soft* real-time sporadic activities should be executed as soon as possible, to obtain a good average response time.

Optional parts cannot be considered as a part of the mandatory ones, nor can they be executed at the same priority level. The execution of optional parts will be determined by the availability of remaining time and the demand for soft sporadic load.

Several approaches to include optional computations in real-time systems can be found in the literature. In one of the first works along these lines (Liu et al., 1991), a task model to incorporate imprecise computation was defined. A refinement of this model was developed by Audsley (Audsley et al., 1991), where the process model was decomposed into three tasks: a mandatory task, a final task and an optional task. From the scheduling point of view, each task has a fixed assigned priority. Based on the original process attributes, the approach consists of calculating the temporal attributes for the task set to be guaranteed. After that, the priority is assigned using a deadline monotonic approach, and a test allows the task-set schedulability to be known. In García–Fornes et al. (1991), the optional activities are controlled by an optional server that can be executed at any priority level using the available slack time (Lehoczky and Ramos–Thuel, 1992; Davies et al., 1993).

Optional parts are relevant if AI techniques are used in the control computation. A survey of this topic may be found in Albertos et al. (1997a).

5. Final remarks

One of the goals of this paper is to show up the interaction between control algorithms and their implementation, if the time constraint is a strong one. The way in which the control is performed determines the algorithm to be used. These constraints can be summarised as

1. The time interval between the information sampling and the control updating. This must be as short as possible, and the range of variation should be also limited. If it is known, the control algorithms can be adapted to take it into account.
2. Scheduling of multiple tasks. This requires adequate computing time.

A set of tools for control-system design and task schedulability, providing time constraint guarantees, have been presented, as a first step in the analysis and design of modern complex distributed control systems.

Some of the issues discussed here may be solved in an alternative way by the use of additional specific hardware, such as multiprocessors, parallel processing, or digital signal processor units, although this reduces the flexibility and slightly increases the cost. However, if the operating conditions are well defined, and mainly if the final purpose is to design an embedded system, hardware solutions should be considered.

References

Albertos, P. (1990). Block multirate input–output model for sampled-data control systems. *IEEE Trans. Automat Control*, *35*(9), 1085–1088.

Albertos, P. (1991). Input–output model for unconventional smapled-data control system. In *Computer Aided Systems Theory*, (pp. 614–625). F. Pichler, R. Moreno-Diaz (Eds.) Berlin: Springer.

Albertos, P., Salt, J., & Tornero, J. (1996). Dual-rate adaptive control. *Automatic*, *32*(7), 1027–1030.

Albertos, P., Crespo, A. Picó, J., & Navarro, J.L. (1997a). Some issues on AI techniques in RT process control. *IFAC Workshop on Automation in the Steel Industry*, Kyongju, Korea (pp. 1–12).

Albertos, P., Sanchis, R., & Sala, A. (1997b). Scarce data operating conditions. *IFAC Symp. on System Identification*, Fukuoka, Japan (pp. 463–474).

Astrom, K., & Wittenmark, B. (1984). *Computer controlled systems*. NJ, Englewood Cliffs: Prentice-Hall.

Audsley N.C. (1991). The deadline monotonic scheduling. Real-Time Systems Research Group, Department of Computer Science, University of York, UK, Report No. YCS146.

Audsley, N.C., Burns, A., Richardson, M.F., & Wellings, A.J. (1991). Incorporating unbounded algorithms into predictable real-time systems. Real-Time Systems Research Group. Department of Computer Science. University of York, UK, Report No. RTRG/91/102.

Berg, M.C., Amit, N., & Powell, J.D. (1988). Multirate digital control systems. *IEEE Trans. Automat. Control*, *33*, 1139–1150.

Burns, A. (1993). Preemptive Priority Based Scheduling: an Appropriate Engineering Approach Real-Time Systems Research Group. Dept. of Computer Science, University of York, UK, Report No. YCS214.

Chen, C.T. (1984). *Linear Systems Theory and Design* (pp. 561–563). New York: Holt, Rinehart & Winston.

Davis, R.I., Tindell, K.W., & Burns, A. (1993). Scheduling slack time in fixed priority preemptive systems. *Proc. Real-Time Systems Symp.* (pp. 222–231), Raleigh-Durham, North Carolina, 1–3 December, Silverspring, MD: IEEE Computer Society Press.

García-Fornes A., Crespo, A., & Botti, V. (1995). Adding hard real-time tasks to artificial intelligence environments. *Workshop on Real-time Programming*, Fort Lauderdale, Florida.

Gonzalez Harbour, M., Klein, M.H., & Lehoczky. J.P. (1991). Fixed priority scheduling of periodic tasks with varying execution priority. *IEEE Real-Time Systems Symp.*

Isermann, R. (1989). *Digital control systems.* Berlin: Springer.

Kalman, R.E. (1964). When is a linear control system optimal. *J. Basic. Engng. 86*, 51–60.

Klein, M.H., Ralya, T., Pollak, B., Obenza, R., González Harbour, M. (1993). A practitioner's handbook for real-time analysis: Guide to rate monotonic analysis for real-time systems. Dodrecht: Kluwer.

Lehoczky, J.P., Sha, L., & Ding, Y. (1989). The rate monotonic scheduling algorithm: exact characterisation and average case behavior. *Proc. IEEE Real-Time Systems Symp.*

Lehoczky, J., & Ramos-Thuel, S. (1992). An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. *Proc. Real-Time Systems Symp.* (pp. 110–123), Phoenix, Arizona, 2–4 December, Silversprings, HD: IEEE Computer Society Press.

Liu, C.L., Layland, J.W. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*. *20*, 46–61.

Liu, J.W.S., Lin, K.J.L., Shih, W.K., Yu, A.C., Chung, J.Y., & Zhao, W. (1991). Algorithms for scheduling imprecise computations. *IEEE Comput*. *24*, 58–68.

Mita, T. (1985). Optimal digital feedback control systems counting computation time of control laws. *IEEE Trans. Automat. Control. AC-30*(6), 542–548.

Salt, J., & Albertos, P. (1990). Digital regulators redesign with irregular sampling. *IFAC World Congress*, Tallin.

Salt, J., Albertos, P., & Tornero, J. (1993). Modeling of non-conventional sampled-data systems. *Proc. 2nd IEEE Conf. Control Applications.* Vancouver, pp. 631–635.

Voulgaris, P., & Bamieh, B. (1993). Optimal H and H2 control of hybrid multirate systems. *Systems Control Lett*., *20*, 249–261.