Jitter Compensation for Real-Time Control Systems

Pau Martí and Josep M. Fuertes
Automatic Control Dept.
Univ. Politècnica de Catalunya
Barcelona, Spain
{pmarti,pepf}@esaii.upc.es

Gerhard Fohler
Dept. of Computer Engineering
Mälardalen University
Västerås, Sweden
gerhard.fohler@mdh.se

Krithi Ramamritham
Computer Science and Engineering Dept.
IIT Bombay, India
krithi@cse.iitb.ac.in

Abstract

In this paper, we first identify the potential violations of control assumptions inherent in standard real-time scheduling approaches (because of the presence of jitters) that causes degradation in control performance and may even lead to instability. We then develop practical approaches founded on control theory to deal with these violations. Our approach is based on the notion of compensations wherein controller parameters are adjusted at runtime for the presence of jitters. Through time and memory overhead analysis, and by elaborating on the implementation details, we characterize when off-line and on-line compensations are feasible. Our experimental results confirm that our approach does compensate for the degraded control performance when EDF and FPS algorithms are used for scheduling the control tasks. Our compensation approach provides us another advantage that leads to better schedulability of control tasks. This derives from the potential to derive more flexible timing constraints, beyond periods and deadlines necessary to apply EDF and FPS.

Overall, our approach provides guarantees offline that the control system will be stable at runtime -- if temporal requirements are met at runtime -- even when actual execution patterns are not known beforehand. With our approach, we can address the problems due to (a) sampling jitters, (b) varying delays between sampling and actuation, or (c) both – not addressable using traditional EDF and FPS based scheduling, or by previous real-time and control integration approaches.

1. Introduction

In control theory, sampling and actuation are generally assumed to be synchronous and periodic, and a highly deterministic timing in task executions is assumed [1]. Specifically, consider the three main parts of a control loop (see Figure 1): sampling, control computation, and actuation. Firstly, sampling should be performed at the same sampling instant every period, secondly, control computation should start and finish quickly after the sample is available, and thirdly, actuation should occur immediately after the control computation, or at a fixed instant after the sampling. Moreover, these three actions are assumed to be instantaneous. However, this is impossible in practice given that the computations take time and may have to contend with other computations for processing and other resources.

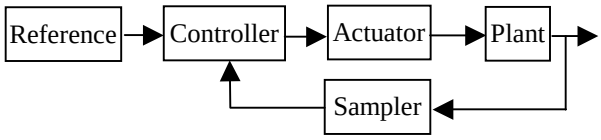


Figure 1. Control loop

When a control algorithm is executed by a task (performing the three actions sequentially) or by a set of subtasks (where each task performs one or more parts of a control loop) in a multitasking real-time system, these assumptions are not met, as scheduling algorithms introduce various forms of jitter to each task instance execution. These jitters can be characterized as:

Sampling Jitter: time intervals between consecutive sampling points may not be constant (even assuming insignificant sampling-actuation delays).

Sampling-Actuation Delays: even if sampling occurs at regular intervals, there could be a delay between when a sample arrives and when the actuation response occurs after the completion of the control computation. This can be due to start-time delays in the control computations. The problem illustrated above is exacerbated due to varying execution times of the control computation.

Sampling Jitter and Sampling-Actuation Delays: this is a combination of the previous two problems and is caused by varying sampling intervals, delays in the start of control computations, non-negligible execution times, and preemptions during the control computations, which in turn can lead to variable actuation times.

These jitters cause control performance degradation and even instability [12]. In addition, the scheduling

algorithm may over-constrain the system when trying to fulfill the stringent timing constraints that control theory mandates, resulting in poor schedulability. Classically, these issues have been treated using either control theory or real-time scheduling theory.

In this paper, we show how these problems can be addressed using a combination of control theoretic and scheduling principles so that control systems can exploit new (and more flexible) scheduling approaches and scheduling approaches can take advantage of control systems properties. Specifically, *we show that combining offline schedulability and control analysis with online scheduling and online control compensation we obtain better schedulability and better control performance*.

Our approaches address the practical problems posed by sampling jitters and varying execution times:

- Traditional EDF [11] and FPS [18] based scheduling approaches don't address these issues. We do, by adjusting the schedule, taking advantage of (a) control properties and (b) the flexibility offered by the compensation approach.
- Real-time and control integration approaches have not examined these issues in their generality to provide better control and schedulability. By combining online compensations along with offline analysis and online scheduling, we are able to provide a solution that handles both sampling and actuation jitters.

Specifically, we identify and discuss the main issues stemming from control in real-time systems that the scheduler must address, and present the control compensation approach as a solution.

The compensation approach is used for compensating the degradation that both irregular sampling (due to sampling jitter) and irregular actuation (due to sampling-actuation delays, which can include varying execution times) causes for the control system response.

This technique was originally suggested as an ad-hoc technique for PID (Proportional, Integral and Derivative) controller design in [21, 2 and 5] and no formal approach was presented. We not only extend the applicability of the compensation technique to deal with both sampling jitter and sampling-actuation delays but also examine its formal as well as practical underpinnings (in particular space and time overheads) for general state space models. The compensation approach affords us the possibility of relaxing the strict periodicity and deadline requirements; we demonstrate how to obtain new flexible timing constraints that can be exploited to improve schedulability.

Overall, we offer a novel approach to real-time and control systems integration that (i) compensates for control degradation due to jitters, (ii) can guarantee stability and (iii) can lead to better schedulability.

The rest of the paper is organized as follows. In Section 2 we discuss control systems and list the types of jitters resulting from real-time scheduling. In Section 3 we present our compensation approach for the three critical jitter types. Section 4 discusses implementation aspects and presents solutions to time and memory problems. In Section 5 we consider the demands placed by our method on real-time scheduling and propose new, flexible timing constraints to exploit the flexibility afforded by our method. The effectiveness of our approaches is illustrated in Section 6 via control simulations. Section 7 summarizes the paper.

2. Impact of Scheduling on Control Performance

2.1. Discrete Control Systems

Broadly speaking, computer-based control systems can be designed following two methods: discretization of a continuous-time design or discrete time design. In both cases, the final controller, obtained using a suitable controller design strategy, must meet the specified closed loop system performance requirements taking into account the dynamics of the process that is controlled. In the end, the controller is a computation that will be executed at every sampling period *h*. This controller is characterized by several design parameters that are highly dependent on the sampling period *h* used in the design stage.

Two important points must be noted. First, if the sampling period is changed and the controller has to be redesigned, the amount of recalculations (overhead) will vary depending on both the design method and the controller design strategy used. Secondly, the selection of the sampling period *h* [1] is determined by the desired performance of the closed loop system and the dynamics of the process that is controlled. An accepted rule-of-thumb is that the sampling frequency should be 4 to 20 times the system's cut-off frequency. This means that the sampling period, traditionally understood as a fixed timing constraint for real-time scheduling, can take a specific value within a specified range.

2.2. Example – Inverted Pendulum

Henceforth we will use an inverted pendulum mounted on a motor driven cart to illustrate different results of the paper. A sketch of this system is shown in Figure 2.

The control problem can be stated as follows: the inverted pendulum (of length l and mass m) can only swing in a vertical plane parallel to the direction of the cart (of mass M), where g is the gravity. To balance the

pendulum, the cart is pushed back and forth on a track of limited length. Balancing fails when the inclination of the pendulum exceeds preset limits, or when the cart hits the stops at the end of the track. The aim is to find a controller to balance the inverted pendulum, preventing it from failing, and bring the cart to the center of the track. The force applied to the cart provides the controlling action (u), calculated according to the actual angle (θ) and position (x). A linear time invariant state space approximation of the inverted pendulum, used for the control design can be found in [13].

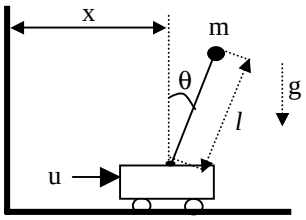


Figure 2. Inverted pendulum

For the sake of simplicity, we will focus only on the angle. Therefore, the goal of our controller is to maintain the desired vertical position of the inverted pendulum at all the times. It can be easily seen that the open-loop system is unstable.

Although we use an inverted pendulum as example, our compensation approach can be applied for linear and linearised non-linear systems. Physical systems are inherently non-linear. However, in many systems, if the system signals do not vary over too wide a range, the system responds in a linear manner. Consequently, even though we deal with non-linear systems, in order to design the control law, the usual procedure that we follow is to work with a linearised approximation model (if not already linear) of the system, concerning the functional parameters. This is the case of the inverted pendulum example we use.

2.3. Types of Jitters

As mentioned in the introduction, realistic implementation of control loops violates the control loop execution assumptions due to sampling jitter, sampling-actuation delays, or both combined. Overall, we have six different cases to consider (see table 1), which in turn can be reduced to three for analysis.

1. Sampling jitter, with negligible controller execution time: time intervals between consecutive sampling points (h_k) are not constant, and this results in sampling jitter.

2. Sampling-actuation delays with strictly periodic sampling, with negligible controller execution time: samples are taken at periodic times (via interrupts, for example). Control computations suffer from start time jitters. As a consequence, actuation instants are not strictly periodic, and this results in varying sampling-actuation delays (τ_k).

3. Actuation at the next sampling point with strictly periodic sampling, with negligible controller execution time: samples are taken at periodic times and actuation is performed at the beginning of the next period. Control computations suffer from start time jitters.

4. Sampling jitter with significant controller execution times: time intervals between consecutive sampling points (h_k) are not constant, which results in sampling jitter. In addition, control computations suffer from start time jitters and may have varying execution times. As a consequence, actuation instants are not strictly periodic, which results in varying **sampling-actuation delays** (τ_k).

5. Actuation jitter with strictly periodic sampling with significant controller execution times, which results in **sampling-actuation delays**: samples are taken at periodic times (via interrupts, for example). Control computations suffer from start time jitters and may have varying execution times. As a consequence, actuation instants are not strictly periodic, which may produce varying sampling-actuation delays (τ_k).

6. Actuation in the next sampling point with strictly periodic sampling, with significant execution times: samples are taken at periodic times and actuation is performed at the beginning of the next period. Control computations may suffer from start time jitters and may have varying execution times.

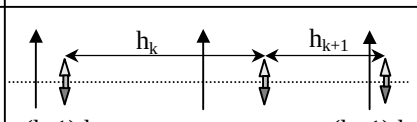
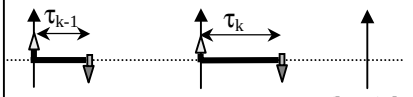
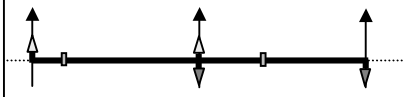
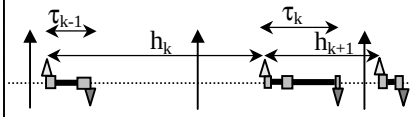
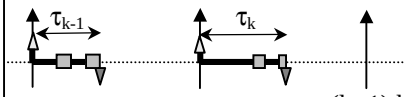
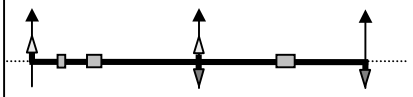
In table 1, for each case we provide a sketch and we derive the main sources of problems that scheduling may introduce in the implementation of a control loop.

Given this case analysis, in the rest of the paper we will focus on cases 1, 2, and 4, which are exactly the ones discussed in the introduction: in case 1 sampling jitter is present, violating the constant sampling assumption. In the same way, in case 2, sampling-actuation delays occur, violating the equidistant actuation assumption. Finally, in case 4, both sampling jitter and sampling-actuation delays occur (and made worse due to varying execution times), which is a combination of the previous two problems.

Notice that in case 5, execution time variability exacerbates the sampling-actuation delay problem. However, from a control viewpoint, it can be modeled as case 2, and it is already included in case 4 for analysis.

Finally, cases 3 and 6 will be no longer discussed in this paper because from a control point of view, they are

not violating the constant sampling or equidistant actuation assumptions. It has to be said that case 6 is violating the assumption that the control computation should start and finish soon after the sample is available. However, if the control computations are generated using the appropriate control model [20], actuation will be performed at the beginning of the next period and control performance will not be affected.

Case	Sketch[1]	S[2]	SA[3]	SP[4]
1		x	-	h_k
2		-	x	τ_k
3		-	-	-
4		x	x	h_k,τ_k
5		-	x	τ_k
6		-	-	-

[1] Δ, □ and ▽ denote sampling time, controller execution and actuation time
[2] S stands for sampling jitter
[3] SA stands for sampling-actuation delays
[4] SP stands for source of problems

Table 1. Jitters summary

3. A Practical Approach to Compensate for the Jitters

The main idea behind the compensation approach is to adjust at runtime the controller parameters at each control task instance execution to account for both the sampling and actuation jitters. By doing the online compensation at runtime, each control task instance can be allowed to execute at any pre-fixed time instant within a sampling interval (h_k). The sampling period variability and the set of values at which any control task instance can start its execution are determined by an offline control analysis that includes a stability analysis [13]. In the end, this implies that sampling and actuation will be performed as follows:

- Sampling: $h_{min} \leq h_k \leq h_{max}$, where h_{min} has to be larger than 1/20 * period equivalent to the systems cut-off frequency and h_{max} has to be smaller than ¼ *the period equivalent to the systems cut-off frequency.
- Actuation: $0 \leq \tau_k \leq h_{k+1}$, where the minimum sampling-actuation delay will be zero (no delay) and its maximum will correspond to the next sampling instant.

In the rest of this section, for each of the three cases of jitters, we analyze what the problem is, which control method applies, what the compensation approach solves, what is needed from the schedule and the implementation details that must be considered.

3.1. Sampling Jitter (case 1)

Time intervals (h_k) between consecutive sampling instants are not constant. If the schedule cannot provide a constant sampling period, the control computation can solve the problem by adjusting the controller parameters according to the h_k at each task instance execution, using the irregular sampling discrete time system model with varying time delays depicted in (1), with τ_k=0 for all k-instance execution. Knowing h_k by time measurements or provided by the scheduler (by adequate offline scheduling analysis), online parameter adjustment is possible.

3.2. Sampling-Actuation Delays (case 2)

Even if sampling occurs at the same point in time, there could be a variable delay (τ_k) between when a sample arrives and when the actuation response occurs following the instantaneous control computation. If the schedule cannot provide equidistant actuation instants, the control computation can solve the problem by adjusting the controller parameters according to τ_k at each task instance execution, using the irregular sampling discrete time system model with varying time delays depicted in (1), with h_k=*h* for all k-instance execution. Knowing τ_k by time measurements or provided by the scheduler (after an appropriate offline scheduling analysis), online parameter adjustment is possible.

3.3. Sampling Jitter and Sampling-Actuation Delays (case 4)

This is a combination of the previous two problems and is caused by varying sampling intervals (h_k), delays in the start of control computations, non-negligible execution times, and preemptions during the control computations, which in turn can lead to variable actuation times. Therefore, varying sampling-actuation delays (τ_k) also appear. If the schedule cannot provide a constant sampling period and equidistant actuation instants, the control computation can solve the problem by adjusting the controller parameters according to h_k and τ_k at each task instance execution, the irregular sampling discrete time system model with varying time delays depicted in (1).

In this case, apart from knowing h_k by time measurements or provided by the scheduler (after an appropriate offline scheduling analysis), we need to know τ_k. Therefore, the on-line scheduler, at runtime, at the beginning of each control task instance execution, must give assurances about the time the actuation will be performed, through adequate offline and online scheduling analysis. Knowing h_k and τ_k, runtime parameter adjustment is possible.

$$\begin{aligned} x(\bar{h}_{k+1}) &= \Phi(h_k)x(\bar{h}_k) + \Gamma_0(h_k,\tau_k)u(\bar{h}_k) + \Gamma_1(h_k,\tau_k)u(\bar{h}_{k-1}) \\ y(\bar{h}_k) &= Cx(\bar{h}_k) + Du(\bar{h}_k) \\ u(\bar{h}_k) &= -L(h_k,\tau_k)x(\bar{h}_k) \\ \bar{h}_k &= \sum_0^k h_k \end{aligned} \quad (1)$$

Due to space limitations, the reader is referred to [13] for an extensive explanation of the control formulation introduced in (1).

3.4. Jitters Summary

Although from a theoretical control point of view, the compensation approach discussed above compensates for the degradation that the controlled system suffers due to jitters, its full applicability in a real-time system also needs to be investigated with respect to its implementation cost (computational overhead and memory requirements) and the availability of the necessary information to recalculate the controller parameters when it is needed (information availability).

With respect to the implementation cost, at each control task instance execution, the controller parameters must be updated according to the actual jitters. Two strategies may apply: runtime or offline calculations.

If the controller parameter adjustment is performed by online extra calculations according to actual jitters, the introduced computational overhead will depend on the control design method and controller design strategy that is being used. If the computational overhead is not negligible, the controller parameter adjustment can be performed online by accessing offline pre-calculated look-up tables. These tables will contain the necessary parameters to allow the control computation to compensate for sampling jitter and/or the sampling-actuation delays that may appear at runtime. In this case, the memory requirements to store these tables must be assessed. We will estimate the size of the tables, which depends on both the design method and the controller design strategy.

With respect to information availability, the implementation of the different control strategies used in the compensation approach mandates that h_k and/or τ_k (for the runtime calculations) must be known at the beginning of each task instance execution. That means that whether runtime or offline calculation approach is used in the jitter compensation, the offline scheduling along with the online scheduling must provide this information at the right time when no time measurements are possible or available.

3.5. Related Work

The state-of-the-art in the field of control and scheduling is represented by [4]. Many fundamental issues in implementing real-time control applications in distributed control systems are discussed in [19].

Some recent research has focused on the jitter problem itself using specific scheduling-based solutions [6, 3, 9]. However, even after modifying the scheduling algorithm, in these approaches, jitter is not completely eliminated. So, if the control design does not take into account the jitters, however small they may be, degradation in the control system performance can still occur. What we propose is to accept the jitter that the scheduling algorithm is bound to introduce and to compensate for it at runtime in the controller design so as to minimise the system degradation that would otherwise occur.

The optimization of control system performance subject to schedulability has also been treated in [16] and [15]. However, these approaches, based purely on offline optimization, do not take into account the runtime effects (jitters) of the scheduled task instances on the control performance. In addition, unlike our approach, no flexibility is allowed at run time.

Runtime control performance and schedulability optimization is treated in [8, 7, 14 and 17]. In these approaches, the major goal has been to adapt at runtime the properties of the schedule, modifying the scheduling algorithm in order to improve schedulability and optimize the control performance. However, no degradation due to jitters is taken into account, even though tasks are prone to jitters in the proposed scheduling algorithms.

In summary, existing approaches have not considered the deleterious effects of scheduling on control system stability even though they have attempted quite successfully to optimize control system performance through the optimized selection of periods, by designing special purpose task models, or scheduling algorithms.

4. Implementation of Compensation

4.1. Controller Parameter Adjustment

In this section, we will show how to implement the control computation incorporating the compensation approach. We first consider the usual control computation and the computation compensating for sampling jitter using online calculations.

For example, for a discretization of a continuous time designed PID controller, at each execution of a PID algorithm (Figure 3, top), the usual computations involve the calculation of the three actions (p_k, i_k, d_k) according to the current error (e_k) in order to obtain the output (u_k). Notice that the integral (i_k) and the derivative (d_k) actions depend on the sampling period *h*. Figure 3 (bottom, in bold), shows what is needed in the PID algorithm in order to be able to compensate for the sampling jitter (h_k), that is, to obtain h_k at each instance execution and to use this value in the rest of the calculations.

```
PID
{   read_inputs (y_k, r_k);
    e_k = r_k - y_k;
    u_k = calculate_output (p_k(e_k), i_k(h,e_k), d_k(h,e_k));
    write_output (u_k);
}
```

```
Compensated PID
{   read_inputs (y_k, r_k);
    e_k=r_k-y_k
    obtain(h_k)
    u_k = calculate_output (p_k(e_k),i_k(h_k,e_k),d_k(h_k,e_k));
    write_output (u_k);
}
```

Figure 3. PID algorithms

However, for a discrete time design and state feedback design strategy, the implementation of the control computation involves more calculations. At each execution of a general state feedback controller (Figure 4, top), the usual computation involves the calculation of output (u_k) according to the state (x_k) and gain matrix L, apart from updating the state for the next controller execution according to the current state and the closed loop matrix (Φ_c). Notice that the gain matrix and the closed loop matrix that depend on the sampling period *h* are fixed parameters of the controller algorithm, calculated at the design stage, because *h* is supposed to remain constant at run time. In Figure 4 bottom (in bold), we see that the extra calculations in the readjusted controller compensating for sampling jitter are, apart from calculating the h_k, to recalculate the discretization of the system model ($\Phi(h_k)$, $\Gamma(h_k)$), the gain matrix ($L(h_k)$), and the closed loop matrix ($\Phi_c(h_k)$).

```
State Feedback Controller
{   read_input (y_k);
    u_k = calculate_output (x_k, -L(h));
    write_output (u_k);
    x_{k+1} = update_state (x_k, Φ_cl(h));
}
```

```
Compensated State Feedback Controller
{   read_input (y_k);
    obtain(h_k)
    calculate (Φ(h_k), Γ(h_k), L(h_k), Φ_cl(h_k);
    u_k = calculate_output (x_k, -L(h_k));
    write_output (u_k);
    x_{k+1} = update state (x_k, Φ_cl(h_k), y_k);
}
```

Figure 4. State feedback controllers

4.2. Computational Overhead

In the previous section, we have seen that the extra calculations of the compensated PID or the compensated State Feedback Controller depend on the design method and control strategy. For the PID, the computational overhead is insignificant because the only new computation is *obtain(h_k)*. However, for the state feedback controller, the computational overhead may be significant. Therefore, we have to precisely evaluate it, and we do so now, for the inverted pendulum example.

To control the angle of the inverted pendulum, we have obtained a state feedback controller by the discrete time pole placement observer approach using Ackerman's formula [1]. In this case, we estimate the computational overhead of the controller performing the extra-calculation of the compensation approach to be $O(n^4)$, where *n* is the closed loop system matrix dimension. For the example of the inverted pendulum (4x4 matrix) and a simplified model (2x2 matrix), the approximate numbers of flops, obtained via simulations for each control task execution, with and without the extra-calculation necessary for sampling jitter compensation are detailed in table 2. This table points out that the overhead of extra calculations may be too high (sometimes, orders of magnitudes higher), depending on the processor speed.

	No. of Flops (without extra calculations)	No. of Flops (with compensation)
simplified pendulum	25	250
pendulum example	60	2000

Table 2. Computational overhead

In summary, if the computational overhead, that can be assessed offline, is insignificant, runtime calculation of controller parameters is a feasible approach. However, if the computational overhead is not negligible, the offline calculation approach, taking advantage of the look-up tables, must be considered. However, we must determine the memory requirements to store these tables.

This is done next.

4.3. Memory Requirements

From the three cases we have analyzed, we can derive the size of the tables.

These tables will store the controller parameters. Depending on the type of jitter the controller will compensate for, h_k will be an input parameter (if compensating for sampling jitter), and for each h_k, τ_k will be a second input parameter (if compensating for sampling-actuation delays). The worst case assumption is to provide for the full range to h_k and τ_k. Knowing that the sampling interval and the delay is a multiple of the clock tick size, the size of the table can be described by:

$(h_{max} - h_{min})*h_{max}*$clkticksize * size(controller parameters)

Notice that in the expression above, the size of a table for compensating only for sampling jitter will not have the term h_{max} and for compensating only for sampling-actuation delays will not have the term $(h_{max} - h_{min})$.

In table 3, we can see the size of the look-up table for the inverted pendulum for each of the three cases, if h_k and τ_k are allowed to take 100 different values within their respective ranges.

	Size(table)
Sampling jitter	8 Kb
Sampling-actuation delays	8 Kb
Sampling jitter and sampling actuation delays	64 Kb

Table 3. Memory requirements

From this analysis, we can conclude that these tables with the offline calculations are small enough to be stored in any micro-controller's RAM.

5. Timing Constraints for RT Scheduling

We will now discuss how the temporal constraints imposed by our jitter compensation approach can be exploited for improving real-time schedulability. First, we address fixed timing constraints as demanded by standard scheduling schemes. Then we show how novel, flexible timing constraints can be used to fully exploit the flexibility provided by our approach to improve schedulability.

5.1. Fixed Timing Constraints

Standard scheduling schemes are based on fixed timing constraints such as periods and deadlines. Here, *fixed* means that a single value for a constraint holds for all instances of a task. Classically, in control, task periods and deadlines are selected as follows:

- *Period:* After an appropriate control analysis, the value of the real-time period is chosen from a range of suitable possible sampling periods.
- *Deadline*: The completion of the actuation task has to occur at the next sampling time (if deadline = period) or a fixed finishing time (deadline < period or deadline > period).

Note that to apply the compensation approach an upper bound on the completion time of the task is required rather than the exact finishing time (recall that in subsections 3.2. and 3.3 these requirements have been already specified). This imposes additional demands on scheduling. In a standard scheduling scheme, this can be enforced, e.g., by non-preemptive execution, or setting a deadline equal to start time plus execution time. In both cases, however, the variability of the execution time can still produce jitter. Furthermore, the strict scheduling requirement can impair schedulability.

The selection of fixed timing parameters for a task has to be based on worst-case assumptions about load scenarios, task phasing, etc. That is, should the load situation be because of a single instance, the timing constraints to meet this worst-case demand have to be imposed on all instances. This reduces schedulability and may even render the tasks unschedulable although all demands may be met from a control perspective. The approach we describe next enables the derivation and setting of timing constraints on a per instance basis, adjusting the task instance timing constraints to the situation faced by that instance. Thus, the timing constraints may vary from instance to instance, but the overall goal of control stability is ensured.

5.2. Flexible Timing Constraints

Our flexible timing method does not set specific values for the timing constraints. Rather, it provides ranges and their combinations to choose from, taking into account, e.g., schedulability of other tasks. In addition, at runtime, the resulting jitter and variations are compensated for as described in the previous section. Thus, our methods provide more flexibility than can be expressed by fixed timing constraints.

We now define the following constraints, for task instances:

- *Instance separation:* The interval between the start of two consecutive instances of a sampling task, S^i and S^{i+1} is limited by the range for sampling intervals: $h_{min} \leq | st(S^{i+1}) - st(S^i) | \leq h_{max}$.
- *Delay:* The interval from the start of sampling to the finishing time of an actuator task is limited by the computation delay: $0 \leq | ft(A) - st(S) | \leq h_{k+1}$. The actual sampling-actuation delay for a particular instance has to be known at the start of the control computation instance for the application of the appropriate compensation. That is, the delay constraint, while flexible, has to be kept such that the actuation task finishes *at* rather than *before the* specified time.

In both cases, the actual values have to be chosen from the set of values we provided compensation for.

Note that while similar for subsequent instances, these constraints differ from the fixed ones: using period T, the i^{th} instance of a task can start at i*T, whereas with the constraints here it can start at $\sum_{j=1,\ldots,i-1} h_j$. Since the h_j can vary, the instance start times will be different.

Instead of simply trying to guarantee an instance based on these constraints, our compensation approach provides the scheduler with the flexibility to select the constraints as well. If a set of constraints cannot be met given the schedule, the scheduler can pick another set of constraints to find a feasible schedule for. Our method provides the possible values to be used and ensures control stability by compensation. Section 6 presents an example.

Note that we do not propose a scheduling approach, rather a new set of flexible timing requirements. At this point, we have used an offline schedule construction approach but are currently investigating new scheduling schemes to handle these new types of constraints.

6. Example and simulations

In this section, we show the effectiveness of the compensation approach, firstly, from a control point of view, and secondly, from a schedulability point of view. All the simulations were carried out with the simulator presented in [10].

6.1. Fixed Timing Constraints

For case 4 (combination of sampling jitter and varying sampling-actuation delays), we show in Figure 5 the possible effects (degradation) on the control performance if the control computation is executing the state feedback controller we designed for the inverted pendulum and scheduled by RMA and EDF (top). We will also show how the compensated state feedback controller compensates for the degradation introduced by the different type of jitters (bottom).

We show in table 4 the details of task 1 and task 2 that were added in order to introduce the jitters into the control task.

	Task1	Task2	Control task
T	60ms	70ms	80ms
C	10ms	10ms	1ms

Table 4. Task set

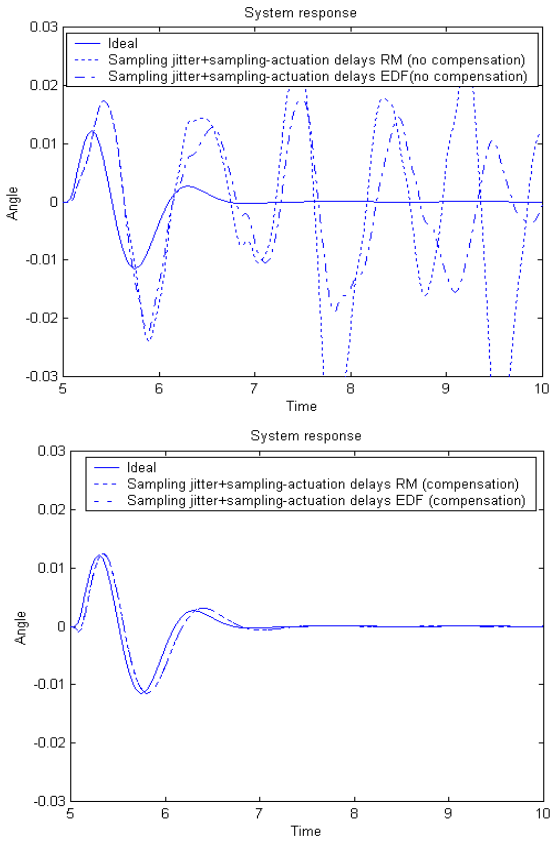


Figure 5. Jitter degrading effects (top) and its compensation (bottom)

6.2. Flexible Timing Constraints

We have studied how flexible timing constraints can improve schedulability of task sets. Now we present an example that cannot be scheduled using fixed timing constraints. By exploiting flexible timing constraints, however, we can construct a schedule that meets control demands.

Consider the task set shown in table 5. As before, tasks T1 and T2 were added in order to introduce the jitters to the control task Cr.

	T	C	DL	Offset
Task T1	100ms	60ms		
Task T2	200ms	20ms	20ms	
Control task Cr	h_k	20ms	20ms	40ms

Table 5. Task set

Suppose the possible values for h_k are 60, 80, or 100ms. Consider scheduling using fixed timing constraints, a schedule for which is depicted below for 400ms. Note that this schedule is executed repeatedly. (Boxes mark periods of tasks, executions are shaded).

For illustrative purposes an h_k value of 80ms has been chosen as the period for Cr. Therefore in this example, h_k for all instances of the control task Cr is set to 80ms. It is obvious that both Cr and T2 need to execute between 200ms and 220ms to meet their respective deadlines, which is not possible. Hence this task set is not schedulable, as shown in Figure 6.

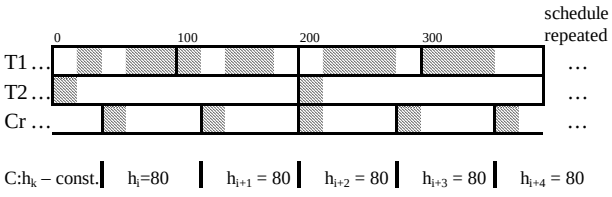


Figure 6. Not feasible schedule

Now, consider associating flexible timing constraints with Cr.

Instead of a selecting only one fixed h_k value for Cr, flexible timing constraints allow us to choose a specific h_k value for each instance. As in the schedule above if we set h_i=80ms, it will cause a scheduling conflict with T2. Instead, we choose h_{i+1} = 60ms and so Cr finishes before T2 starts. Then, we choose h_{i+2} = 100ms and h_{i+3} = h_{i+4} = 80ms. Using these h_k values instead of a fixed period, the task set can be scheduled, as shown in Figure 7. Still, as Figure 8 shows, because task executions are based on the use of compensations, which take into account the specific h_k value used, stability is maintained – even though task instances have different timing constraints.

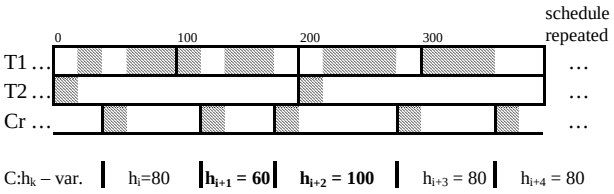


Figure 7. Feasible schedule

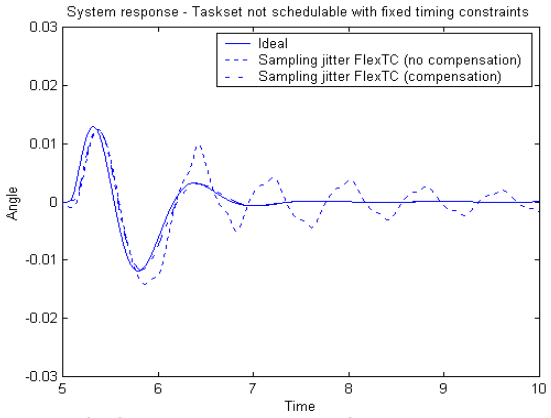


Figure 8. System response of a task scheduled using flexible timing constraints

Note that by applying flexible constraints, the task set, which is not schedulable using fixed constraints, can be scheduled to meet the control demands.

7. Summary

In this paper, we proposed a new approach for real-time scheduling of control systems by compensating for sampling jitter and sampling-actuation delays through the adjustment of controller parameters. We discussed the impact of scheduling methods on control performance and identified the critical types of violations that can occur due to sampling jitter, sampling-actuation delays, and both combined. We presented an approach using control theory to calculate the adjusted controller parameters to compensate for these violations. These parameter adjustments can be made (a) online -- if runtime overheads are acceptable, or (b) offline -- via table lookups at run time, if the overheads are not negligible. We discussed practical issues, in particular computation costs and memory needs, to make the above decision. Furthermore, we proposed new, flexible timing constraints to fully exploit the flexibility of our methods to improve schedulability. These constraints are defined on a per control task instance basis, as opposed to fixed values, such as periods and deadlines, applicable to all instances –

as assumed by standard scheduling schemes such as EDF and FPS.

Simulation studies show the effectiveness of our approach for standard real-time scheduling schemes. By accepting the jitter inherent in standard scheduling schemes and including it in control analysis our method can effectively maintain control stability. Our results also show how the flexible timing constraints can be used to provide control stability in real-time control tasks that are not schedulable with fixed timing constraints.

We have applied an offline scheduling scheme to exploit the flexible timing constraints to enhance schedulability. As part of our work planned for the future, we are investigating online as well as combined offline-online algorithms.

8. Acknowledgments

This work has received support from the Spanish CICYT project ref. DPI2000-1760-C03-01.

9. References

[1] K.J. Åström and B. Wittenmark. *Computer Controlled Systems. Third edition*. Prentice Hall. 1997.

[2] P. Albertos and J. Salt. "Digital Regulators Redesign with Irregular Sampling", *11th IFAC World Congress* (Preprints), vol 8, pp 157-161, 1990.

[3] P. Albertos, A. Crespo, I. Ripoll, M. Vallés and P. Balbastre "RT control scheduling to reduce control performance degrading". *39th IEEE Conference on Decision and Control*. Sydney (Australia), December 12-15, 2000.

[4] K-E. Årzen, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson and L. Sha "Integrated Control and Scheduling". Research report ISSN 0820-5316. Dept. Automatic Control, Lund Institute of Technology, 1999.

[5] K.-E. Årzen, A. Cervin, J. Eker and L. Sha. "An Introduction to Control and Scheduling Co-Design", *39th IEEE Conference on Decision and Control, Sydney*, Australia, December 12-15, 2000.

[6] S. Baruah, G. Buttazzo, S. Gorinsky and G. Lipari, "Scheduling Periodic Tasks Systems to Minimize Output Jitter" *Proc. International Conference on Real-Time Computing Systems and Applications*, IEEE Computer Society Press. pp 62-69, Hong Kong, December 1999.

[7] G. Buttazzo, G. Lipari and L. Abeni, "Elastic Task Model for Adaptive Rate Control*" IEEE Real-Time Systems Symposium, Madrid*, Spain, December 1998.

[8] M. Caccamo, G. Buttazzo and L. Sha, "Elastic Feedback Control", *IEEE Proc. 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, pp. 121-128, June 2000.

[9] A. Cervin "Improved Scheduling of Control Tasks" in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, England, June 1999.

[10] J. Eker and A. Cervin, "A Matlab Toolbox for Real-Time and Control Systems Co-Design" In Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications, Hong Kong, China, December 1999.

[11] Liu, C. and J. Layland "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". *J.ACM*, **20**, 46-61. (1973).

[12] P. Marti, R. Villa, J.M. Fuertes and G. Fohler, "On Real-Time Control Tasks Schedulability", *European Control Conference*, Porto, Portugal, September, 2001.

[13] P. Marti, R. Villa, J.M. Fuertes and G. Fohler, "Stability of On-line Compensated Real-Time Scheduled Control Tasks", *IFAC Conference on New Technologies for Computer Control*, Hong Kong, China, November 2001.

[14] L.Palopoli, L.Abeni, F.Conticelli, M. Di Natale, G. Buttazzo, "Real-Time control system analysis: an integrated approach." In Proc. of the Real-Time System Symposium, Orlando, Florida, November 2000.

[15] H.Rehbinder, M. Sanfridson "Integration of Off-Line Scheduling and Optimal Control", *12th Euromicro Conference on Real-Time Systems*, Sweden, June 2000.

[16] D. Seto, J.P. Lehoczky, L. Sha and D.G. Shin, "On Task Schedulability in Real-Time Control Systems". *RT Systems Symposium, 17th IEEE*. p 13-21, 1996.

[17] L. Sha, X. Liu, M. Caccamo and G. Buttazzo, "Online Control Optimization Using Load Driven Scheduling", *39th IEEE Conference on Decision and Control, Sydney*, Australia, December 12-15, 2000.

[18] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. Real-Time Systems The International Journal of Time-Critical Computing, 6:133--151, 1994. 17.

[19] M. Törngren, "Modelling and Design of Distributed Real-Time Control Applications". Phd Thesis. ISRN KTH/MMK-95/7-SE. Dept. of Machine Elements, KTH, Sweden, 1995.

[20] R.J Vaccaro, *Digital control. A state-sapace approach.* McGraw-Hill 1995.

[21] B. Wittenmark and K.J. Åström. "Simple Self-tuning Controllers". In Unbehauen, Ed. Methods and Applications in Adaptive Control, number 24 in Lecture Notes in Control and Information Sciences, pp 21-29. Springer-Verlag, Berlin, FRG, 1980.