Real-Time Control Design for Flexible Scheduling using Jitter Margin

Moris Behnam, Damir Isovic
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Sweden
{moris.behnam,damir.isovic}@mdh.se

June 5, 2007

Abstract

Real-time control algorithms are designed based on the characteristics of the controlled plants and they require good performance without delays. However, digital control implementation typically introduces delays and jitters due to insufficient CPU processing power and the limitations of the real-time scheduling method used. This can degrade the system performance or even make it unstable.

In this paper we propose an integrated approach for control design and real-time scheduling, suitable for both discrete-time and continuous-time controllers. It guarantees system performance by accepting a certain minimum value of jitter for control tasks and feasibly schedules them together with other tasks in the system. Results from comparison with other approaches from real-time and control theory domains underline the effectiveness of our method.

1 Introduction

Most of hard real-time control applications are developed in two stages; control design and computer implementation stage [6]. In the control design stage, the control engineers use design theories to evaluate control models that make the overall system fulfill the desired performance. In the computer implementation stage, the designed control model is converted into a set of periodic *control tasks* with hard deadlines, which are then allocated to the processor according to some real-time scheduling policy.

However, the real-time implementation of a controller may lead to non-constant time intervals between consecutive invocations of the control task, caused e.g., by pre-emptive, priority-based scheduling where tasks can interrupt and delay each others execution. As a consequence, real-time scheduling may introduce *jitter* in sampling and actuation parts of the control task, which have not been considered at the control design stage. This can degrade the control performance and may cause system instability.

One approach to solve the problem of jitter in control tasks is to simply use a dedicated processor for each control loop and to assign the highest priority to the corresponding control task, among all other tasks executing on the processor. However, having one processor per control loop is not a cost-effective solution, and in many cases, not even possible because of physical space limitations on the target system, e.g., a small embedded device. Instead, control tasks must coexists with other hard real-time tasks on the same processor, tolerating pre-emptions in a predictable way.

A method to minimize the effects of actuation jitter based on Deadline Monotonic scheduling policy [4] has been proposed in [12]. It splits the control task into data acquisition task, algorithm evaluation task and actuation task, each with the same period as the

original control task but with different priorities and offsets. The actuation task is assigned the highest priority to minimize the actuation jitter.

Another approach presented in [2] minimizes actuation jitter by splitting the control task into two parts: the calculating output part, which should be executed as soon as possible to generate the control action, and the updating part which can be executed with some delay. The execution time of the control task is reduced so the jitter and the time delay will be minimized as well. However, there will be some jitter left after modification, influencing the system stability.

In [9], the effects of jitter are eliminated by sending the control signal in the beginning of next sample after measuring the input signal and calculating control action. The issue with this approach is that it introduces unnecessary high time delay.

An approach to compensate the effect of jitter at runtime has been presented in [13]. A runtime compensator adjusts the controller parameters to eliminate the effect of sampling and actuation jitter. While being flexible, this method introduces extra runtime overhead.

The methods above are focusing on handling jitter *after* the design of the controller, i.e., in the scheduling algorithm, neglecting one important issue: the control system design can affect the real-time scheduling in a negative way. The period of the control task is determined during the control design stage, without taking into account other tasks in the system and its possible negative impact on the schedulability of the system. Hence, we believe that it is required to *integrate* the design of control system and real-time scheduling to eliminate the effect of jitter while satisfying the desired control performance requirements.

In our work, we adopt *jitter margin* approach [1] to obtain the acceptable jitter sets that ensures control system performance, and develop an integrated real-time control design and scheduling method that keeps the desired control system performance within required range, without violating the deadlines of other hard real-time tasks. The design objective is to select a sampling interval for the control tasks such that the minimum control performance is satisfied, while the scheduling objective is to provide for coexistence with other tasks in the system while minimizing the jitter of the control tasks. We use offline real-time scheduling to construct a real-time schedule that fulfills the evaluated constraints from jitter margin calculations. We evaluate several possible solutions according to allowed jitter intervals and select the best one.

It is important to notice that the original jitter margin approach tries to minimize the performance degradation as much as the used scheduling policy allows, while in our method, we consider the required performance as a design parameter to be used as an input for the scheduler. This way, we are working actively towards the desired performance from the beginning rather than adapting to the scheduler specifics. Besides, jitter margin uses apparent phase margin as performance factor when selecting timing constraints for control tasks, while in our approach we use both apparent phase margin and disturbance amplification factor. Moreover, the jitter margin approach presented in [1] is suitable only for continuous controller design, while our method include also discrete time LQG controllers.

Control system applications usually prefer offline scheduling because the execution of all tasks is predetermined. However, the price to pay is flexibility, i.e., it is difficult to include dynamic tasks arrivals with not completely timing properties at design time. Hence, our real-time scheduling approach is *joint* offline and online scheduling. We use offline scheduling for control and non-control, periodic hard real-time tasks, and introduce the flexibility into the system by performing online scheduling of dynamic task arrivals, such as aperiodic and sporadic tasks with unknown parameters at the design time. In our previous work [5, 7] we have showed a method which allows online scheduling of dynamic tasks on top of an offline schedule, called slot shifting. In this work, we focus on offline scheduling of control and non-control, hard real-time tasks, and present a novel scheduling algorithm that fully exploit the proposed control design method. It is based on simulated annealing algorithm, i.e., a global optimization technique that attempts to find the lowest point in an energy landscape. Our proposed scheduling method finds an offline schedule that fulfills the evaluated constraints from jitter margin calculations.

Integration of the offline scheduling and the control design algorithm depends on the type of control design, hence we have developed, implemented and analyzed our method for both continuous-time and discrete LQG controller design. We compared the results with other scheduling methods used in the control systems that are sensitive to jitter in actuation, and showed that our approach performs better.

The rest of the paper is organized as follows: Section 2 presents our design method for real-time control based on jitter margin. In Section 3 we show how our design method is applied when designing continuous-time and discrete-time controllers. In Section 4 we show how to schedule the proposed design by using simulated annealing, which we extend in Section 5 with online scheduling to provide flexibility for dynamic task arrivals. In Section 6 we present the evaluation of proposed methods, followed by Section 7 which concludes the paper.

2 Design Method

The sampling and the actuation jitter of a control task, caused by the real-time scheduling, can degrade the system performance from the control point of view and, hence, they should be minimized. A simple method that eliminates the jitter in both sampling and actuation is to use offline scheduling. We can instruct the scheduler to make sure that the sampling and the actuation code of the control task are fixed for each task invocation, while the execution of the control algorithm can vary between the fixed parts. This assumes that the sampling part in a control task is executed at the beginning of the task while the actuation part is executed at the end of control task after control calculation. Assume, for example, two

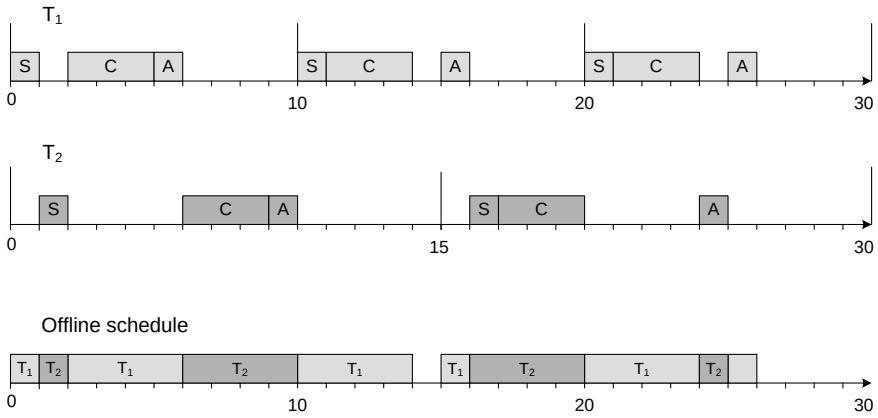

Figure 1: Example simple offline schedule without jitter

control tasks $T_1(10, 5)$ and $T_2(15, 5)$, where the first parameter is the period of the task and the second one is the worst-case execution time (WCET). The objective is to schedule these tasks such that the jitter in sampling and actuation is equal to zero. An offline schedule that fulfills the requirements is shown in figure 1 (the part of the control task where the sampling takes place is denoted by S, the control algorithm part by C and the actuation by A). We can see from the figure that the time distance between two consecutive samplings and actuation for both control tasks is constant. On the other hand, the control algorithm executes with non-constant time separation, but it does not cause any sampling or actuation jitter. The only effect of the offline scheduling on the control tasks is constant time delay, i.e., T_1 will have constant time actuation delay 6 while T_2 will have constant time actuation delay 10. If the system is sensitive to this delay, we can use the knowledge obtained from the scheduling to redesign the controller taking into account the delay, or we can limit the value of time delay in offline scheduling to ensure required performance if it is hard to

redesign the controller.

However, there are two drawbacks using the method described above. First, offline scheduling can be constructed manually for simple cases (few tasks and small LCM of tasks periods), but for complex systems it becomes very hard. Second, since the jitter value should be zero, finding feasible scheduling becomes very hard. Therefore we propose a new design method for real-time control that provides for more flexibility in underlying real-time scheduling.

The main idea of our method is to accept a certain value of jitter for each control task, such that the system performance will be kept within required range, and hence increase the possibility to find a feasible schedule. We start by shortly describing the jitter margin approach and then we show how it can be used in our design methods for continuous-time controller and discrete-time LQG design.

2.1 Jitter Margin

Assuming that the actuation signal in a control task is sent at the end of task's execution, the time delay for the task will be bounded by the interval [BCRT, WCRT], where BCRT and WCRT denote the best-case respective the worst-case response time of the task. The *minimum time delay*, L, and the *jitter*, J, are then given by:

$$\begin{aligned} L &= BCRT \\ J &= WCRT - BCRT \end{aligned}$$

Jitter margin approach [1] can be defined as the largest number $J_m(L)$ for which closed-loop stability is guaranteed for any time varying delay in the interval $[L, L + J_m(L)]$. The largest value of L that guarantee system stability is called *delay margin* and denoted L_m.

Jitter margin has the following properties:

1. $J_m(L) = 0, L \geq L_m$
2. $J_m(L) \leq L_m \forall L$
3. $L + J_m(L)$ is an increasing function of L

For example, figure 2 shows the jitter margin for a DC servo control system. By specifying the value of the minimum time delay L=0.008 seconds, we can evaluate the jitter margin as $J_m(0.008) = 0.0038$ and if L=0.01 $J_m(0.01) = 0.0019$. The delay margin for this example is $L_m = 0.01475$. Please see [1] for details on jitter margin.

2.2 Performance factors

We assume that the minimum required performance is provided (by control engineers) in terms of performance degradation, either as degradation of apparent phase margin or disturbance amplification (original jitter margin uses only apparent phase margin).

The *apparent phase margin*, $\hat{\phi}_m$, is the largest number for which the closed loop stability is guaranteed for any time varying delay in the interval $[L + \hat{\phi}_m/\omega_c, L + J + \hat{\phi}_m/\omega_c]$, where ω_c is crossover frequency for constant time delay L. The jitter value that guarantees certain $\hat{\phi}_m$ is evaluated as:

$$J = J_m(L + \hat{\phi}_m/\omega_c) \tag{1}$$

If $\hat{\phi}_m < 0$ then the system stability is not guaranteed while for $\hat{\phi}_m > 0$ the system stability is guaranteed.

Here is an example: assume that we want to evaluate the maximum jitter for DC servo controlled by LQG controller that is designed for constant delay of 5, such that L=5, $\hat{\phi}_m = 20^o$ and $\omega_c = 57.8$. The jitter margin for this example is shown in figure 2, and it is equal to:

$$J_m(L + \hat{\phi}_m/\omega_c) = J_m(5ms + 20^o/57.8\ radius) = 1.3ms$$

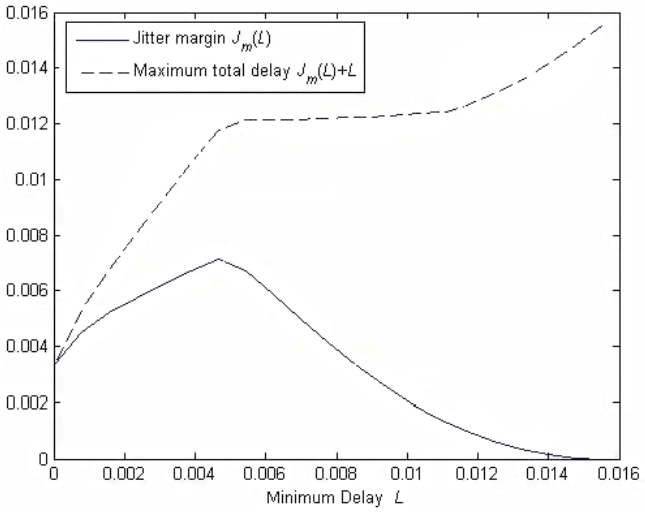

Figure 2: Example jitter margin for LQG

Another performance factor we use is *disturbance amplification*, i.e., performance degradation bound as a measure of sensitivity gain which effect on disturbance gain. The impact of the delay for continuous-time system on performance is the scaling factor of $1/(1-\gamma)$ on the disturbance gain, where γ is the total loop gain and it can be calculated using jitter margin calculations , see [3] for details.

If $\gamma > 1$, then the system stability is not guaranteed, while if the γ=0 that means there is no degradation in the system. Hence, it is required to make γ as small as possible to ensure that the disturbance gain will not increase and thus increase the effect of the disturbances. For example, figure 3 shows performance degradation bound as a function of jitter J and constant delay L for DC servo controlled by PID controller. Assume it is required that the external disturbances do not increase by more than 1.3 times. Then, from scaling factor equation above we get the value of γ=0.2. This implies that the values of L and J are limited by the curve 0.2 in figure 3.

2.3 Integrated Control Design and Real-Time Scheduling approach

By applying jitter margin calculation to achieve the given performance degradation factor, we get the maximum allowable jitter as a function of the minimum time delay L for each control task.

Figure 4 shows the curve that represents jitter as a function of required performance and minimum time delay L, the required apparent phase margin $\hat{\phi}_m = 40^o$, the sampling interval h=50 and the task execution time c= 0.002s. We can use any point from the curve and schedule the control task offline such that the actuation delay is limited to $[L, L + J_m(L)]$. This means that we enforce the best-case and the worst-case response time of control task to be:

$$\begin{aligned} BCRT &\geq L \\ WCRT &\leq L + J_m(L) \end{aligned}$$

Since we assume that the actuation signal is sent at the end of the control task, the minimum value of L should not be less than the execution time of the task, so we have to evaluate the *best-case execution time* (BCET) for the control tasks to obtain accurate BCRT. However, finding a scheduling policy that can limit the response time of the control task to $[L, L + J_m(L)]$ is not easy and sometimes even impossible using online scheduling. On the other

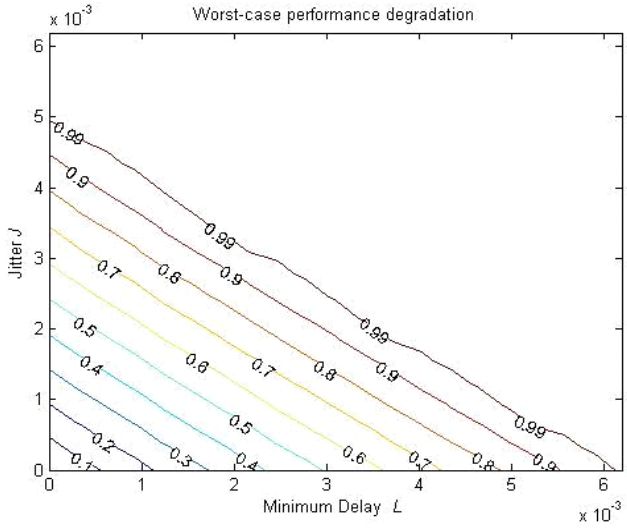

Figure 3: Worst-case performance degradation bound for DC servo controlled by PID controller.

hand, if we use offline scheduling, we are allowed to use more options to find a feasible schedule that can limit the response time of the control task.

Using BCET in offline scheduling imposes additional scheduling restrictions, which in this case must ensure that the actuation should not be sent before the selected value of L. One way to simplify this problem is to split the control task into two tasks; the *sampling-control task* and the *actuation task*, and to release the actuation task assuming BCET=WCET, i.e., before $L + J(L)$. This will increase the average time delay for actuation, because in the case of BCET the actuation can be sent earlier, but it has an advantage of not needing to calculate BCET, i.e., we can use WCET instead as most of the scheduling algorithms do.

This approach will provide more flexibility in scheduling since only the actuation task execution is limited according to the jitter margin. The sampling-control task can be released at the beginning of the period and it can execute anywhere from the start of the period until the start of the actuation task. The only constraint is that it should complete before the actuation task starts, which is easy to achieve by using the precedence relation between the actuation task and the sampling-control task.

After providing jitter margin curves for all control tasks, we use offline scheduling to schedule them together with the non-control hard real-time tasks. The only restriction for non-control tasks is that they must be scheduled to complete before assigned deadlines, i.e., they can be flexibly scheduled (and pre-empted) between their release time and the deadline. The restriction is on the control tasks instead, i.e., the actuation task and the sampling-control task, where the actuation task's execution window is decided according to the jitter margin, and sampling-control task execution should be finished before the starting of actuation task.

One of the jitter margin properties is the increasing function of L (see section 2.1), which means that whenever we increase the value of L we get a higher deadline, $D = L + J_m(L)$, but a smaller jitter as shown in figure 4. This makes the scheduling more flexible and increase the possibility to find a feasible schedule because there are more than one accepted deadlines as function of L for each control task and the offline scheduling algorithm can check all these deadlines until getting feasible schedule.

Integration of the offline scheduling and the control design algorithm depends on the type of control design (continuous-time control design, discrete-time control design). We

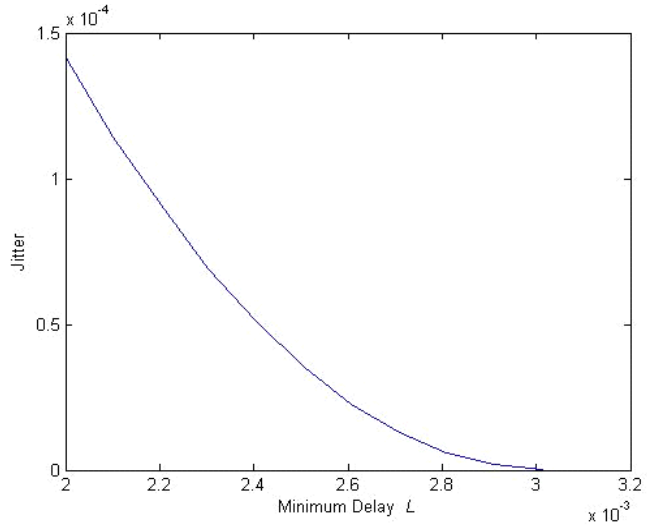

Figure 4: Example jitter margin for servo

will describe the use of our algorithm both with continuous-time controller and discrete-time LQG design.

3 Control Design

Here we show how our design method is applied when designing continuous-time and discrete-time controllers, such that the overall system schedulability is guaranteed.

3.1 Continuous-time controllers design

We consider continuous-time controllers where the only parameter that can be changed is the sampling interval (period of control task). The sampling interval affects both control system performance and the real-time scheduling. Our design method finds the sampling intervals for control tasks such that both the minimum control system performance and a feasible real-time schedule are provided.

There are some criteria for determining an appropriate sampling interval, usually referred as rules-of-thumbs, saying that the appropriate sampling interval has to be within a certain range defined by a given system parameter or its mathematical expression. For example, in [9] the sampling interval h for each controller is selected according to the following condition :

$$h * \omega_b \in [0.2, 0.5] \tag{2}$$

where ω_b is closed-loop bandwidth.

We use the condition above to select an appropriate sampling interval when converting a continuous-time controller into discrete-time control task (needed for computer-based control). We start by choosing a number of h values from the equation above and evaluate their jitter margin according to required performance, expressed in terms of apparent phase margin and/or disturbance amplification factor (as described in subsection 2.2). This will give us the maximum deadline for each control task as a function of h.

We arrange the obtained deadline values in a table, which gives an overview of the effect of changing h on each controller. By doing this, we can evaluate a *region* of possible h values, i.e., a set of h that may give feasible scheduling. This can be done by excluding the values of h that cannot achieve the required performance, i.e., those that produce a negative jitter margin within the defined range of L, $L \in [WCET, L_m]$. Also, we can

exclude all combinations of h values for control tasks that make total system utilization of greater than 1.

The design algorithm for continuous-time controllers is summarized by the following:

Repeat
  for each control task
    /* *select the sampling interval*/
    /* *e.g., with help of eqn 2*/
    h = getSamplingInterval();

    /* *find a set of deadlines D as a function of L*
    that satisfy required performance ($\hat{\phi}_m$ or γ)/
    D(L) = jittermargin(h,control system, performance)

  if (U $\leq$ 1) /* *utilization for all tasks */*
    /* *try to schedule */*
    scheduleFound = offlinescheduling(all tasks)
until scheduleFound *(or give up)*

In the next section, we will present our offline method called from the algorithm above.

3.2 Discrete LQG design

Here we describe our approach with discrete-time controller design of a LQG controller. To get higher performance, the constant time delay that is used in controller design should be as close as possible to the best-case response time for control task. We use the constant time delay as design parameter for the LQG controller. Then, with jitter margin calculations we evaluate maximum deadline for the control task. By putting this information in tables for each task we can use the same algorithm that is described in the previous subsection and instead of changing the sampling interval we change the constant time delay for LQG design.

We continue here with the description of the offline scheduling algorithm used. Full examples of both design algorithms and their usage and evaluation with the offline scheduler will be presented in section 6.

4 Offline Scheduling for Real-Time Control

We use *simulated annealing* in our scheduling algorithm. It is a global optimization technique, which attempts to find the lowest point in an energy landscape. The most important function in the algorithm is the energy function which forms the peaks and the valleys at the surface in the world and this affects how the algorithm can reach a solution. The probability of making jumps between energy levels (neighbour points) is controlled by a control variable C, which can be viewed as the temperature factor in a thermodynamic system. By reducing the value of C during the process the probability of making jumps or moves to higher energy levels becomes smaller and the system will become more and more stable until it freezes at some minimum energy point. Please see the original article [8] for details on simulated annealing.

We force all tasks to execute inside their execution windows. The execution window for a non-control hard task will be from the task's release-time until its deadline. For the actuation task, the execution window is decided by jitter margin calculation $[L, L + J_m]$, while the execution window of sampling-control task is given by the interval [task release-time, $L + J_m$]. However, sampling-control task should finish before starting of

actuation task, i.e., there is a precedence relation between these tasks. The precedence relation can be resolved by either including the violation of the precedence relation in the energy calculation or forcing this relationship during tasks allocation.

4.1 Scheduling algorithm

We start by allocating all tasks randomly within their execution windows, and then we use simulated annealing to resolve collisions. In our approach, the energy calculation for neighbor points is based on the number of collisions between tasks (more than one task executes in the same time slot). First, we randomly select a staring point, a source point P_s. Then, a new point, P_n, is selected from the neighbor-space of P_s. This is done by either a) moving a task or part of a task randomly inside its execution window, b) changing the start time and deadline for actuation task depending on jitter margin results, or c) shaking a task or set of task instances left or right such that it will not violate other tasks, i.e., moving a task to empty time slots inside its execution window. The new point is then evaluated against the source point, and if its energy is less or equal to the source point, we automatically accept P_n as new source point. The process is repeated until a thermal equilibrium has been reached, i.e., a feasible offline schedule has been found. Here is the pseudo-code:

/ Initialization */*
- Allocate all tasks randomly within their exec. windows
- Chose a random starting source point P_s
- Chose the control variables (starting temperature) C_s
- Define MAXINNERLOOP, MAXOUTERLOOP

*/*Save P_s as the best point seen so far*/*
$P_b^g = P_b^l = P_s$

Repeat
  Repeat
    / calculate the number of collisions*
    *between task instances for source point */*
    E_s = Energy at point P_s

    / reschedule the tasks */*
    Choose a new point P_n, in the neighborhood of P_s
    E_n = Energy at point P_n
    if ($E_n < E_s$) then
      $P_s = P_n$
      $P_b^l = P_n$ */* P_n is new local best point */*
    else if ($E_n == E_s$) then
      $P_s = P_n$
    else
      $x = (E_s - E_n)/C_s$
      if ($e^x \geq random(0,1)$) then
        $P_s = P_n$
  until thermal equilibrium or MAXINNERLOOP
  if ($E_b^l \leq E_b^g$)
    $P_b^g = P_b^l$ */* back to best point */*
    $P_s = P_b^g$
  / set new temperature that is more restrictive*
  for scheduling (less bad points will be accepted)/*

$C_s = F(P_b)$
until thermal equilibrium or MAXOUTERLOOP

To sum up the algorithm, we can say that the inner loop has freedom to reach new and hopefully better points with a lower energy level. The outer loop ensures that the best point seen during the process is not lost during the scheduling process. The counters in both the loops make sure that the process will always terminate at some time, so there is no risk for eternal loops.

5 Flexible Real-time Controll Scheduling

Control system applications usually prefer offline scheduling because the execution of all tasks is predetermined. However, the price to pay is flexibility, i.e., it is difficult to include dynamic tasks arrivals with not completely timing properties at design time. For example, if we have a system where we need to handle aperiodic tasks together with the offline scheduled control tasks, we must provide leeways in the offline schedule to include aperiodic tasks at run-time when they arrive. The problem is that we do not know exact arrival times of the aperiodic tasks a priori. We solve this problem by using our *slot shifting* method.

5.1 Joint Offline and Online Scheduling

Slot shifting [5, 7] is a scheduling method that combines offline and online scheduling. Offline schedules will generally not be tight, i.e., there will be times where resources are unused. It uses online scheduling on the top of an offline schedule to efficiently reclaim those resources, and use it to schedule dynamic tasks, such as aperiodic or sporadic tasks. First, an offline scheduler creates scheduling tables for all tasks with known timing attributes before run-time by constructing sequences of task executions. The resulting offline schedule consist of independent tasks with start times and deadlines. Second, created offline schedule is divided into a set of disjoint executions *intervals*, I, where each interval is associated with the execution of all tasks with the same deadline. Since we know what is the total execution demand in each interval (i.e., the sum of the executions of all belonging tasks), the amount of free resources per interval is also known, so called *spare capacity*, sc. Third, the execution of offline scheduled tasks is shifted at run-time to accomodate for dynamic task arrivals.

The presented design method works efficiently using the slot shifting algorithm because the actuation task accepts a certain value of jitter, which means that it can be shifted to provide space for aperiodic task. The sampling-control task can also be shifted because it is released at the beginning of the period and it should end before the actuation. Non-control hard tasks can be executed anywhere within assigned slot shifting intervals. The information that should be provided to the slot shifting algorithm is the offline schedule with release times and deadlines for each task.

Beside providing a simple and efficient way to access the amount of available CPU time at runtime, our method also provides for efficient reclamation of unused resources: if a task uses less CPU resources as assigned by the scheduler, those are simply added to the corresponding interval.

5.2 Example

Figure 5-a depicts an example with a controller task that has been split into the sampling-control task, T_{SC}, and the actuator tasks, T_A, and three other non-control hard real-time tasks T_{H1}, T_{H2}, and T_{H3}. We show two consecutive instances of the controller tasks (that has period of 9), denoted as T^0_{SC} and T^1_{SC} (the same is valid for T_A).

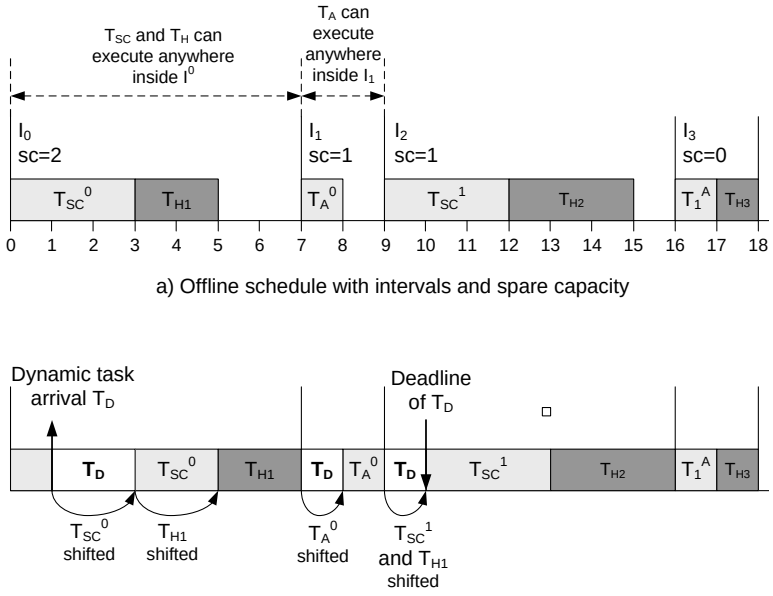

Figure 5: Flexible scheduling of offline tasks to accommodate for online arrivals

The offline scheduler presented in section 4 will create a scheduling table with release times and deadlines for each of the tasks. The offline preparation mechanism of slot shifting will take the offline schedule and create a set of intervals such as the tasks with the same deadline will constitute and belong to the same interval. Assume for example that T_{SC}^0 has release time 0 and the deadline 7 (which is also the release time of T_A^0, while its deadline is 9). Let one of the non-control tasks, T_{H1}, have the same deadline, hence T_{SC}^0 and T_{H1} will belong to the same interval, i.e., I_0 that has start time 0 and the end time 7, i.e., the deadline of T_{SC}^0 and T_{H1}.

Spare capacities are calculated for each interval. For example, it is 2 for I_0, indicating that there is some free resources to include dynamic tasks in that interval, while the spare capacity of I_3 is zero, which means that no dynamic tasks can be accepted to execute in I_3.

Scheduled task can be executed flexibly within their intervals in order to accommodate for tasks that arrive at run-time, as illustrated in figure 5-b. Assume a dynamic task T_D (e.g., a firm aperiodic) arrives at run-time at time 1 and it must be completed at time 10. Then, the execution of the offline scheduled tasks in the arrival interval I_0 will be shifted (postponed) to accommodate for T_D. We can se from the figure that the remaining portion of T_{SC}^0 and entire T_{H1} will be executed later than originally scheduled, as long as the spare capacity in the interval is greater than 0 (for each executed tick of T_D, the spare capacity will be decreased). When the spare capacity reaches zero, this means that offline tasks must be executed, otherwise they will miss their deadlines. This occurs at time 3, hence T_D will continue to run in forthcoming intervals, i.e., I_1 and I_2.

6 Method Evaluation

In this section, we present two examples how to use our method and evaluate the results against two real-time scheduling methods and one method from the control theory domain.

6.1 Continuous-time controller

To show how our method can be applied for continuous-time systems, and for easier comparison, we use the same example as in [1], with three continuous-time plants $P_1(s)$, $P_2(s)$ and $P_3(s)$. The controllers for the three plants are designed using LQG continuous-time controller design method $C_1(s)$, $C_2(s)$ and $C_3(s)$, and implemented on a single CPU.

$$P_1(s) = \frac{8 * 10^5}{s^2 + 1000s}, \quad C_1(s) = \frac{6857s + 8 * 10^6}{s^2 + 4736s + 1.071 * 10^7}$$

$$P_2(s) = \frac{4 * 10^4}{s^2 - 4 * 10^4}, \quad C_2(s) = \frac{1.796 * 10^5 s + 4.811 * 10^7}{s^2 + 5331s + 1.417 * 10^7}$$

$$P_3(s) = \frac{5 * 10^7}{s^3 + 100s^2 + 2.5 * 10^5 s},$$

$$C_3(s) = \frac{2.385 * 10^5 s^2 + 4.032 * 10^7 s + 3.788 * 10^{10}}{s^3 + 10^4 s^2 + 3.316 * 10^7 s + 4.059 * 10^{10}}$$

The design objective in this example is to find sampling intervals h for each system that can fulfill the minimum required performance. The required performance factor is presented in term of apparent phase margin, given by the second column of the table 1. The worst case execution time for all controllers is 0.15 ms. The third column of the table 1 shows maximum sampling interval that can be selected for each system from equation 2.

System	$\hat{\phi}_m$	h_{max} (ms)
$P_1(s)$, $C_1(s)$	50^o	0.86
$P_2(s)$, $C_2(s)$	35^o	1.00
$P_3(s)$, $C_3(s)$	40^o	3.00

Table 1: Required minimum performance in terms of apparent phase margin for a Continuous-time controller

By applying the scheduling method presented in section 4 we get the results shown in table 2. We assume that the actuation task execution time is 0.01 s. We can see from the table that the jitters for all systems are very small, which is very hard to achieve if we use online scheduling (as we will show later). The time response for the systems is shown in figures 6, 7 and 8.

System	h	Min. delay	Deadline	Jitter
$P_1(s)$, $C_1(s)$	0.45	0.29	0.30	0.01
$P_2(s)$, $C_2(s)$	0.50	0.20	0.22	0.02
$P_4(s)$, $C_3(s)$	0.60	0.45	0.46	0.01

Table 2: Our integrated approach applied on a Continuous-time controller

We evaluated our method against two different methods: one from the real-time scheduling domain, and one from the control theory domain. First, we compared our results to Rate Monotonic (RM) scheduling method [11]. Selecting the values of h according to h=h_{max}/2 give results shown in table 3. The stability of systems 3 is not guaranteed since the apparent phase margin is less than 0, while system 2 did not meet its minimum requirement.

Then, we compared our method with a method from control theory domain, described in [1]. The results obtained by method [1] are reported in table 4. We can see from the table that the results are better than those obtained by using Rate Monotonic, but still system 2

and system 3 could not meet the minimum requirements (Note that, it is not easy to even get these results using the design algorithm proposed in [1] since it uses iterative procedure to find h values, so the final results depend on initial values of h and gain parameter k that is used in calculating new sampling intervals).

System	h	Jitter	$\hat{\phi}_m$
$P_1(s), C_1(s)$	0.35	0	65^o
$P_2(s), C_2(s)$	0.50	0.15	30^o
$P_3(s), C_3(s)$	1.50	0.30	-10^o

Table 3: Rate Monotonic applied on a Continuous-time controller

System	h	Jitter	$\hat{\phi}_m$
$P_1(s), C_1(s)$	0.45	0	59^o
$P_2(s), C_2(s)$	0.57	0.15	29.17^o
$P_3(s), C_3(s)$	0.60	0.30	32^o

Table 4: Method from [1] applied on a Continuous-time controller

As it can be seen from the tables 2, 3 and 4, our method outperforms both compared approaches.

6.2 DTC - Discrete-time controller

Suppose that we want to control two mechanical servo using LQG discrete controllers on a single CPU:

$$P_1(s) = \frac{1000}{s^2 + s}, \qquad P_2(s) = \frac{1000}{s^2 + s}$$

The design of the controller is based on LQG discrete controller that is presented in [10], which allows using time delay (LQG-delay) as a parameter in the design. The worst-case execution time for both controllers is 2 ms. The sampling interval and the required performance in term of worst-case disturbance amplifications for each system are given in table 5.

System	h (ms)	Disturbance amplitude
System1	5	1.2
System2	6	1.3

Table 5: Required minimum performance terms of disturbance amplification for a DTC

By using our design and scheduling method (as described in sections 2,4), we get the results as shown in table 6. The table shows very small jitter for both systems.

If we compare our results with online scheduling methods such as Rate Monotonic or Earliest Deadline First [11], we can see that it is hard to get the minimum requirements for one or both systems using RM or EDF because of the high jitter value introduced by these methods, see table 7 for EDF results (to show a fair comparison, both systems are designed using LQG controller with constant deadline = minimum delay = 2ms, the time unit in the table is in ms). Once again, out method performs significantly better than all compared approaches.

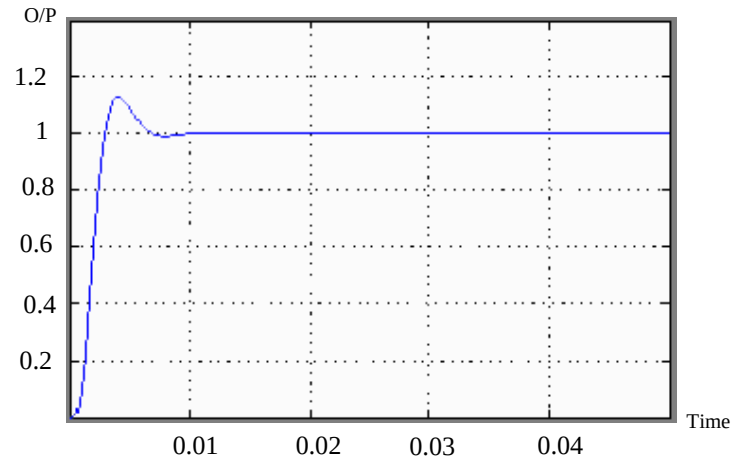

Figure 6: System 1 time response for our method

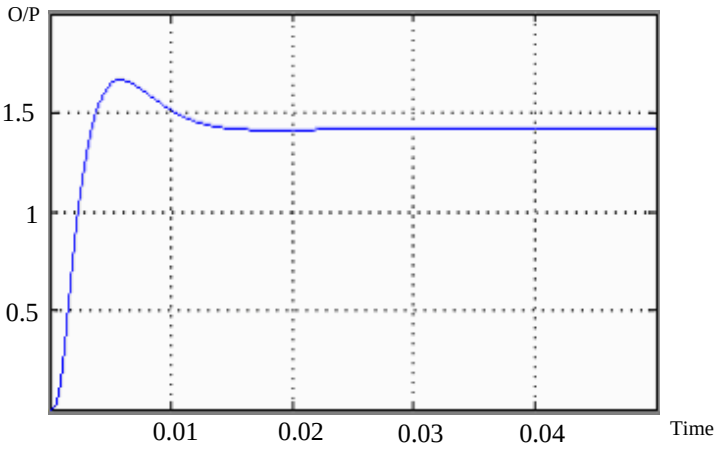

Figure 7: System 2 time response for our method

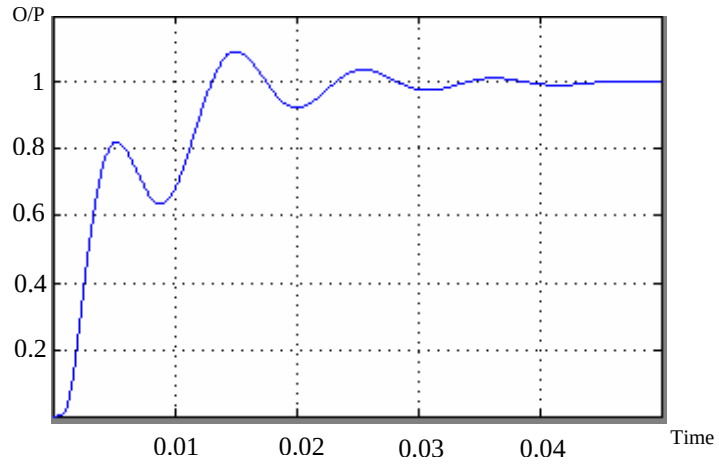

Figure 8: System 3 time response for our method

System	LQG -delay	Min. delay	Deadline	Jitter delay
System1	2	2	2.1	0.1
System2	2	3.7	4	0.3

Table 6: Our method applied on a DTC

System	LQG-delay	Jitter	Disturbance amplitude
System1	2	1	1.3
System2	2	2	1.48

Table 7: EDF applied on a DTC

7 Conclusions

In this paper we presented a method for an integrated real-time control and scheduling co-design, suitable for both continuous-time and discrete-time controllers. The method uses jitter margin approach to obtain allowed range of jitter for control tasks, proposes suitable timing constraints, and flexibly schedules them together with other, non-control, tasks in the system. The design objective is to select a sampling interval for the control tasks such that the minimum control performance is satisfied. The method can use both apparent phase margin and the worst-case disturbance amplifications as a performance factor when selecting appropriate sampling interval. Furthermore, we developed an offline scheduling method that is capable to schedule the control tasks with jitter margin requirements. We split each control task into a sampling-control tasks and an actuation task in order to minimize the actuation jitter, and, at the same time, to allow more flexible scheduling.

We presented two concrete examples of how our method can be used and compared the results with two common scheduling algorithms. We showed that our method satisfies the performance requirements with a significantly smaller actuation jitter for the control tasks. The analysis also shows that the proposed algorithm is more suitable for discrete-time controllers because the high average actuation delay effect can be compensated using the controller design.

References

[1] Cervin A, Lincoln B, Eker J, Årzén K, and Buttazzo G. The jitter margin and its application in the design of real-time control systems. In *Real-Time and Embedded Computing Systems and Applications*, Gothenburg, Sweden, August 2004.

[2] Cervin Anton. *Towards the Integration of Control and RealTime Scheduling Design*. PhD thesis, 2000.

[3] Lincoln B. *Dynamic Programming and Time-Varying Delay Systems*. PhD thesis, 2003.

[4] A. Burns, N.C. Audsley, M.F. Richardson, and A.J. Wellings. Hard real-time scheduling: the deadline monotonic approach. *Proceedings of the IFAC/IFIP Workshop, UK*, 1992.

[5] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *IEEE RTSS*, Pisa, Italy, 1995.

[6] Kang G and Xianzhong C. Computing time delay and its effect on real-time control systems. *IEEE transaction on control systems tech. Vol. 3, NO 2*, June 1995.

[7] D. Isovic and G. Fohler. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *21st IEEE RTSS*, USA, November 2000.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[9] Åström K.J. and Wittenmark B. *Computer-Controlled Systems*. Prentice Hall, 1997.

[10] B. Lincoln and A. Cervin. Jitterbug reference manual. Technical report, Department of Automatic Control, Lund Institute of Technology,Sweden, 2003.

[11] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journnal of the ACM, 20, 1*, Jan. 1973.

[12] Albertos P., Crespo A., I. Ripoll, M. Valls, and P. Balbastre. Real-time control scheduling to reduce control performance degrading. In *39th IEEE Conference on Decision and Control*, Australia, 2000.

[13] Marti P., Fohler G., Ramamritham K., and Fuertes J.M. Jitter compensation for real-time control systems. In *22nd IEEE Real-Time Systems Symposium*, London, UK, December 2001.