Performance-Driven Post-Processing of Control Loop Execution Schedules

SUMANA GHOSH, SOUMYAJIT DEY, and PALLAB DASGUPTA, Indian Institute of Technology, Kharagpur

The increasing demand for mapping diverse embedded features onto shared electronic control units has brought about novel ways to co-design control tasks and their schedules. These techniques replace traditional implementations of control with new methods, such as pattern-based scheduling of control tasks and adaptive sharing of bandwidth among control loops through orchestration of their execution patterns. In the current practice of control design, once the static execution schedule is prepared for control tasks, no further control-related optimization is attempted for improving the control performance. We introduce, for the first time, an algorithmic mechanism that re-engineers a recurrent control task by enforcing switching between multiple control laws, which are designed for compensating the non-uniform gaps between successive executions of the control task. We establish that such post-processing of control task schedules may potentially help in improving the combined control performance of the co-scheduled control loops that are executing on a shared platform.

CCS Concepts: • **Computer systems organization** → **Embedded and cyber-physical systems**; **Embedded software**; • **Software and its engineering** → *Real-time schedulability*;

Additional Key Words and Phrases: Embedded system, task scheduling, loop execution, performance

ACM Reference format:
Sumana Ghosh, Soumyajit Dey, and Pallab Dasgupta. 2020. Performance-Driven Post-Processing of Control Loop Execution Schedules. *ACM Trans. Des. Autom. Electron. Syst.* 26, 2, Article 13 (October 2020), 27 pages.
https://doi.org/10.1145/3421505

1 INTRODUCTION

The traditional development cycle of an embedded software-based control system consists of a controller design phase followed by the implementation of the controller on the target architecture. Depending on the plant's natural response and other control parameters, control engineers choose the sampling period (h) for the controller. The sampling period specifies the periodicity of the software components for that control task, and the software tasks are accordingly scheduled on the target architectural platform, typically using standard real-time scheduling algorithms, such as Earliest Deadline First (EDF) [31].

S. Ghosh is the corresponding author.
The authors wish to thank the IMPRINT grant FMSAFE for partial support of this work.
Authors' addresses: S. Ghosh, S. Dey, and P. Dasgupta, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India 721302; emails: sumana61189@gmail.com, {soumya, pallab}@cse.iitkgp.ac.in.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
1084-4309/2020/10-ART13 $15.00
https://doi.org/10.1145/3421505

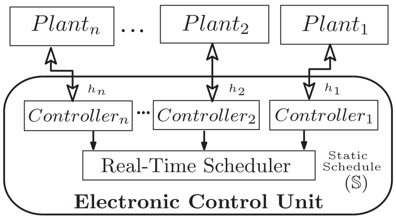


Fig. 1. Embedded Architectural Platform.

The recent trend for scheduling multiple software control loops on shared Electronic Control Units (ECUs) as shown in Figure 1, primarily aims to address cost and connectivity concerns. For example, in the automotive domain, following AUTOSAR [16] recommendations, sharing ECUs among control applications help in simplifying the vehicle's electronic and networking architecture. On the other hand, co-scheduling the control tasks of multiple control loops on a shared ECU offers new challenges and opportunities in terms of choosing how to share the ECU bandwidth. In traditional control design, it is common practice to use sampling rates with sufficient margin in order to compensate for the occasional execution drops that may occur due to non-idealities of the execution platform. For example, due to the delay in receiving sensory inputs in a sampling interval, execution of the software controller may potentially be skipped at that sampling interval. Though such over provisioning of control task execution may work well for single task mapped ECU systems, implementing the idea directly in a multi-task shared ECU setting leads to significant scheduling overhead. This has opened up new paradigms of control task scheduling. For example, while control loops are typically executed at uniform intervals of time, recent research [3, 15, 21, 32, 44] has shown that non-uniform execution of control loops following some (m, k)-firm [38] or weakly-hard constraints [5] can be as effective as uniform execution of the loops in terms of quality of control [21, 32, 44] and resource utilization [3, 15]. The effect of drops on control performance has been studied in the past, but an important difference of such works with this line of work is that in our adopted setting of control task scheduling, the drops are intentionally introduced by the scheduling algorithm. Such intentional *loop skipping*, by inserting empty slots, i.e., instrumented *drops* in a uniform schedule of execution, when performed in a regular manner, establishes the possibility of designing loop execution patterns having competitive control performance. An (instrumented) *drop* in a loop execution pattern represents an instance of skipping the execution of the control loop, thereby making the ECU free in the corresponding time slot. A *drop* in one control loop can be suitably utilized by another control loop.

Figure 2 illustrates the notion of pattern-based execution of the control loops, and the manifestation of drops in loop execution patterns for two control loops having sampling periods of $h_1 = 2$ ms and $h_2 = 3$ms, respectively. The execution times for the control tasks for the two loops are 1ms and 1.5ms, respectively. The down and up arrows are used to indicate the start and end of the task execution interval of the loop. In a window of 12ms, the first loop has 6 execution slots of 1ms each, and the second loop has 4 execution slots of 1.5ms each. The loop execution pattern shown in Figure 2 does not use all these slots (in the figure, the used slots are marked by green and red colors for these two loops, respectively). Instead, the executions of the second and fifth instances of the first loop are skipped, and the first and fourth instances of the second loop are skipped. If we denote a loop execution by '1' and execution drop by '0,' we obtain a loop execution pattern, 101101 for loop-1 and 0110 for loop-2. The figure shows one single instance of both patterns and they are executed recurrently, i.e., the same pattern repeats every 12ms. The use of drops in loop

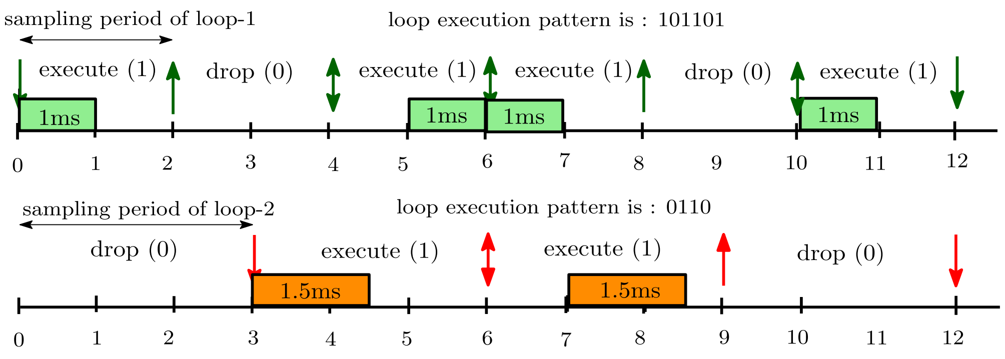


Fig. 2. Pattern-based execution of control loops.

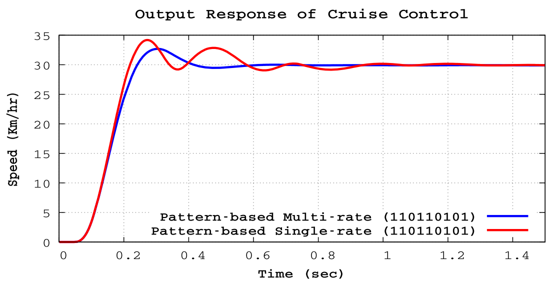


Fig. 3. Pattern-based Execution: Single vs Multi-rate.

execution patterns facilitate the co-scheduling of multiple control loops on a shared ECU, provided that it does not adversely affect the quality of control.

The focus of this article is on post-processing a given set of co-schedulable loop execution patterns to improve the combined control cost of the loops. Given such patterns as input, one can gain prior knowledge regarding the regular positioning of the loop-drops, which can be leveraged in the following two ways, and is the primary focus of this article:

(W1) *The loss of control performance due to insertion of drops can be compensated by modifying the control law in the execution instance preceding one or more drops in the execution pattern.*

This can be observed from Figure 3 where we compare the output response of a cruise control system [39] when (i) the loop is executed following the loop execution pattern, 110110101, with only one control law (red curve) and (ii) we follow the same execution pattern but employ multiple control laws chosen judiciously based on the position of the drops (blue curve). The benefit of applying multiple control laws is clearly evident from the better response of the system in case of option-ii.

(W2) *Drops (or alternatively execution slots) can be traded between control loops to improve the combined control cost of the loops.*

In this article, we present an algorithmic methodology for leveraging the drops in loop execution patterns and post-processing a given set of execution patterns. The proposed methodology has the following two main components which are two main contributory steps in this work:

(1) *The design of a switched controller for a loop execution pattern with drops.* Instead of using the same controller as in the uniform sampling schedule, we design a control task that switches between multiple control laws depending on the number of successive drops between executions. The motivation behind adapting multiple control laws in a

pattern-based execution of controllers is to take care of the performance degradation caused due to loop drops (corresponds to W1 and details are given in Section 4).

(2) *Localized modifications in the given schedule of the control tasks by selectively exchanging computational slots allotted to them.* Note that, a slot exchange between two loops generates new loop execution patterns for those loops. Slot exchanges aim to improve the combined control performance of the loops subject to their schedulability and stability constraints (corresponds to W2 and details are given in Section 5).

In general, most of the existing approaches for embedded control focus either on optimizing the controller designs for performance objectives being oblivious of platform constraints or on the co-design of control and platform parameters. Tracking both the goals, i.e., optimal design of multiple controllers and platform parameters (e.g., execution schedules), is, in general, an intractable problem requiring exhaustive search and massive computational effort [39]. Moreover, such methods limit themselves to the choice of periodic controllers for each control loop. As an alternative, we leverage the *weakly-hard* real-time paradigm [5] of real-time scheduling in the context of control-architecture co-design. This motivates using (m, k)-firm schedules of multiple control loops allowing usage of smaller sampling periods with execution skips instead of designing controllers for individual loops with larger sampling periods for ensuring schedulability. However, the underlying global optimization problem has several parameters to instantiate: (1) choosing the (m, k)-firm patterns for different control loops which are schedulable together and (2) choosing the control gains at each actuation instance of each of the patterns identified. Existing works, either periodic design of controllers that are co-schedulable or freeze the pattern per loop and choose a uniform gain for the same [11, 30, 43].

Given the hardness of the global optimization problem, our proposed post-processing method takes an alternative approach assuming an initial pattern-based co-schedulable control scheduling solution, trying to iteratively improve it by exchanging control execution slots and locally modifying controller gains while ensuring schedulability. We present the first formal exposition of this post-processing problem and the preliminary algorithms towards this. We show that our framework leads to significant gains in terms of a combined quadratic cost function considering well-known case studies.

The article is organized as follows: Section 2 presents related work. Section 3 describes the mathematical foundation of the plant and controller models. Section 4 introduces the formalism for switched loop execution patterns and the underlying theory for applying multi-rate control based on it. Section 5 describes the design problem for switched loop execution patterns and proposes the algorithmic framework for this problem. Case studies and experimental results are given in Section 6. Section 7 presents concluding remarks.

2 RELATED WORK

This article presents a novel formal approach for trading computational slots between control loops resulting in significant gains in the combined control cost. While the proposed notion of post-processing the task execution schedules is new, there exists a rich volume of existing literature on the core components of the proposed approach, namely, the non-uniform execution of the control loops, dynamic sharing of bandwidth among control loops, and use of multi-rate controllers.

The use of a policy for inserting drops in periodic execution of control loops in terms of (m, k)-firm constraints, is introduced in [38]. An (m, k)-firm task model implies that a task will be executed at least m out of any k consecutive sampling intervals, where m and k are two positive integers with $m \leq k$. Such a drop policy is used to reduce the effective load on the processor by carefully discarding execution of control task instances at some sampling intervals. This enables the sharing of ECUs among multiple control components. The research presented in [15], [27], and [44] provides methods for optimizing the platform resources required for ensuring stability by exploiting the ability to drop executions of the control loops. The approaches presented in [3], [20], [21], [32], and [45] follow non-uniform recurrent patterns of loop execution for effective utilization of the processor and/or improved system performance. Moreover, by adaptively changing the execution pattern, one can improve the combined quality of control [21, 44]. Control theoretic analyses are developed in [3], [15], [20], [21], [32], [44], [45], for identifying loop execution patterns with performance and stability guarantees for closed loop systems, whereas in [27], bound on number of drops is decided based only on the scheduling constraints ignoring the control aspect completely. Recently, the concept of convergence rate abstractions of weakly-hard constraints [5] is proposed in [18] that simplifies the stability analysis of weakly-hard systems as is explained using a simple one-dimensional system description. The proposed model also facilitates efficient computation of bounds on the worst-case system state at runtime. On the other hand, formal verification of a general class of weakly-hard systems is proposed in [25]. To manage the complexity of reachability analysis problem of such systems the authors provide relaxation and over-approximation techniques and develop algorithms for verifying the safety of weakly-hard systems.

An automaton-based representation of stable non-uniform scheduling patterns of control tasks guaranteeing the system performance is developed in [3, 45]. The research effort by [32] proposes a control theoretic approach for elucidating the effect of drop-rate on the control performance followed by a scheduling algorithm for generating the static schedule that optimizes the control performance. In [44], a buffer control mechanism is introduced to optimize resource usage by accommodating the dropped execution where in any interval an upper bound of drops are calculated guaranteeing the desired stability. The notion of adaptive sharing of ECU bandwidth in terms of pattern-based recurrent loop execution has been formalized in [21]. In a recurrent execution pattern for a control loop, the delays between successive executions are not uniform, but they follow a recurrent pattern. Adaptive sharing of ECU-bandwidth is enabled by allowing the control loops to trade their execution slots by switching between provably stable loop execution patterns, while guaranteeing the schedulability. A methodology for synthesizing robust loop execution patterns under different input disturbance scenarios is developed in [20]. The proposed methodology also ensures desired energy consumption requirement and co-schedulability of the loops, while optimizing the control performance. But none of these works exploits the advantage of multi-rate controllers for handling the effect of execution drops as we do in this work. On the other hand, in [15], depending on the plant's state information, different (m, k)-firm execution patterns are computed dynamically. In order to minimize the degradation caused by loop drops, the use of multi-rate control schemes are recommended in [15], and [38], but the impact on the combined control performance is not studied in [15] and [38]. In contrast, in this work, the proposed method provides multi-rate control scheme for the loops taking care of the combined control performance. The effect of non-uniform sampling in the design, stability, and control performance of the networked control systems can be found in [6], [19], [26], [30], [36], [37], and [40].

A different line of research focuses on dynamic sharing of computational bandwidth among control loops. Research presented in [9], [10], [11], [14], [23], [33], and [43] propose to manage processor overload by adaptive changes in the sampling rates at runtime and guaranteeing task schedulability in different resource conditions. In particular, Castane et al. [8], Cervin et al. [9], Eker et al. [14], and Henriksson and Cervin [23] solve a schedulability problem where the solution of the underlying optimization problem is a set of sampling periods for which multi-rate controllers are designed accordingly. Aligned with this group of works, research endeavored in [10] and [11] shows that controllers mapped to the same ECU can tradeoff each other's bandwidth on-the-fly in response to the input disturbances to achieve a better overall control performance. However, from a practical perspective, online optimization of sampling rates against control performance metrics is too expensive computationally, and typically infeasible for controllers with high sampling rates. The effort by Marti et al. [33] addresses the problem of dynamic sampling period assignment for handling the reduction of bandwidth due to transient overloads in real-time systems. However, all these works use optimal linear quadratic state feedback controllers for different sampling rates. As shown in [42], control systems designed using the optimal linear quadratic technique may suffer from instability during switching, in spite of the individual controllers being stable. In contrast, our methodology uses the multi-rate controllers based on the LMI-approach provided in [42], such that stability is assured under switching.

There exists a significant amount of work concerning multi-rate control schemes [12, 22, 34, 35]. The usefulness of multi-rate sampling methods for controlling an acceleration control systems is proposed in [34], whereas the stability issue of such multi-rate sampling scheme is addressed in [35]. The benefit of multi-rate control on a pre-configured operating system (e.g., ERCOSek) with a limited set of sampling periods and limited computational power, is demonstrated in [12] and [22]. The proposed approach reduces the processor load, while satisfying the control performance requirement and enabling the integration of more applications into an ECU, thereby saving costs. Platform aware schemes for scheduling embedded controllers have been reported in [46], where one derives asymptotically stable switched multi-rate schedules that maximally utilize the residual bandwidth of the processor. However, these studies do not consider the problem of co-scheduling multiple multi-rate controllers for different plants on the same ECU.

3 CONTROL SYSTEM MODELING

Plant-Control Loop. In this work, we consider a discrete linear time invariant (LTI) plant, $P = (A_p, B_p, C_p)$, with the dynamic equations given as follows:

$$\begin{aligned} x_p[k+1] &= A_p x_p[k] + B_p u[k] \\ y[k] &= C_p x_p[k]. \end{aligned} \tag{1}$$

The vector $x_p[k]$ represents the plant state at the kth sampling interval, for some $k \in \mathbb{N}$, or equivalently at real-time t such that $t = kh$, where h is the sampling interval. The vector $x_p[k]$ represents the plant state at the kth sampling interval, for some $k \in \mathbb{N}$, or equivalently at real-time t such that $t = kh$, where h is the sampling interval. On the other hand, $y[k]$ and $u[k]$ represent the output and the control input, respectively, at that interval. The matrices A_p, B_p, and C_p describe the transition matrix, the input map, and the output map for the plant model, respectively. Let $\Gamma = (A_c, B_c, C_c)$ be the stabilizing feedback controller for P, that senses the plant output y and takes the control decision by adjusting the current values of the control variables in u. The feedback control law is also represented as an LTI system having the following form:

$$\begin{aligned} x_c[k+1] &= A_c x_c[k] + B_c y[k] \\ u[k] &= C_c x_c[k], \end{aligned} \tag{2}$$

where, $x_c[k]$ represents the state of the controller at time $t = kh$. The matrices A_c, B_c, and C_c describe the state transition matrix, the input map, and the output map for the controller Γ, respectively. The closed loop, $\Sigma = \langle P, \Gamma \rangle$, is shown in Figure 4. With $x = [x_p^\top, x_c^\top]^\top$ as the closed loop state vector, the dynamics of the resulting closed loop system, $\Sigma = \langle P, \Gamma \rangle$, is obtained by substituting the values of $y[k]$ and $u[k]$ into the dynamics of plant (Equation (1)) and controller (Equation (2)) respectively:

$$x[k+1] = \begin{bmatrix} A_p & B_p C_c \\ B_c C_p & A_c \end{bmatrix} x[k] = A_1 \, x[k], \tag{3}$$

where A_1 represents the closed loop dynamic matrix for the periodic loop execution.

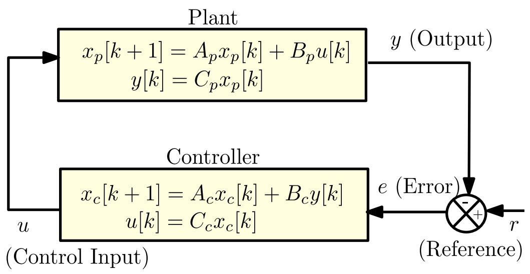


Fig. 4. Closed Loop LTI System.

Drop Scenario. Following our earlier work [21], the notion of *drops* in control loop execution is formalized as follows: Inside a sampling interval $[k, k+1)$, when execution of the control task is dropped, the value of the control variables remain the same and we have $x_c[k+1] = x_c[k]$. In this interval, the closed loop dynamics is therefore:

$$x[k+1] = A_0\, x[k],$$

where the dynamic matrix A_0 is same as A_1 with A_c and B_c being replaced by the identity matrix I, and null matrix, O, respectively, with appropriate dimensions. Hence, the dynamic matrix for the dropped sample is:

$$A_0 = \begin{bmatrix} A_p & B_p C_c \\ O & I \end{bmatrix}.$$

Loop Execution Pattern. For a given control loop $\langle P, \Gamma \rangle$ together with its dynamic matrices $\{A_0, A_1\}$, a *loop execution pattern* is an infinite computation schedule of that control loop, generated by an infinitely repeating finite length string $s \in \{A_0, A_1\}^*$. Therefore, $\forall k \in \mathbb{N}$, the dynamics of the closed loop system, $\langle P, \Gamma \rangle$, is defined as:

$$x[k+1] = s[k\%l]x[k],$$

where l is the length of s. For example, according to the loop execution pattern $s = A_0 A_1 A_0 A_1 A_1$, the closed loop system evolves as:

$$x[5] = A_1 x[4] = A_1 A_1 x[3] = \ldots = A_1 A_1 A_0 A_1 A_0 x[0].$$

For convenience of notation, we express loop execution patterns as binary strings as discussed earlier. For example, $A_0 A_1 A_0 A_1 A_1$ is denoted by the pattern $s = 01011$. Therefore, in a loop execution pattern, "1" represents a loop execution instance, while "0" represents a loop drop instance in that sampling interval.

4 INSTRUMENTING SWITCHED MULTI-RATE CONTROL

In this section, we first introduce the notion of *switched loop execution pattern* of a control loop, which allows stable switching among multiple controller gains depending on the time gap between successive executions of the control loop. The idea of using multiple such gain in a pattern-based execution of controllers is to take care of the performance degradation caused due to loop drops. As the measure of the quality of control of a switched loop execution pattern, we next devise a theory for defining the quadratic cost of a switched loop execution pattern.

The drops in a loop execution pattern can adversely affect the quality of control, even if the stability of the modified controller is guaranteed. This is illustrated in Figure 5 for one of our case studies, which is a cruise control system. The response shown in green is for a uniformly

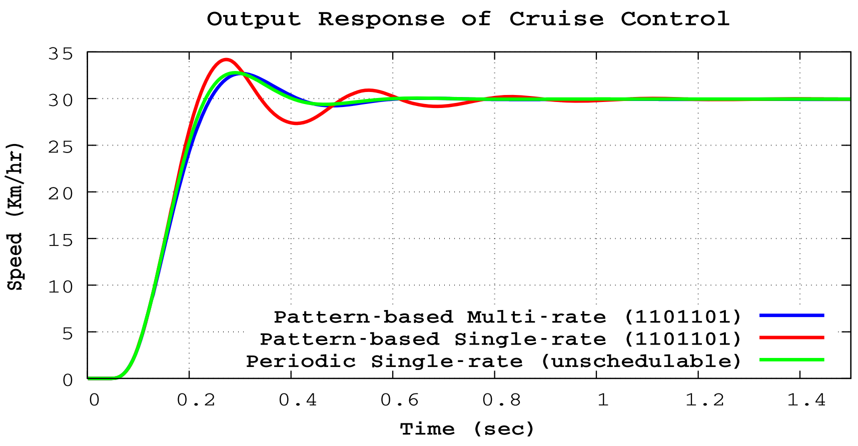


Fig. 5. Compensation for drops using switched multi-rate control.

executed (periodic) controller. The response shown in red is for a modified schedule of execution following the loop execution pattern, $s = 1101101$. Evidently both the settling time and peak overshoot are adversely affected due to the drops. We propose to compensate this degradation by choosing a different control law in the execution instances immediately preceding the drops. The response shown in blue is for the switched controller designed with the proposed approach, and it is evident that this response is as good as the original periodic controller, although it uses fewer computation slots (due to the use of pattern). The reason behind modifying the control law in the execution instance preceding one or more drops is as follows: Consider the loop skipping subsequence 100, where there are two consecutive loop drops following one execution instance. We are given a controller designed for some sampling period, h. Note that a control execution drop excludes the control values from being updated, whereas the plant evolves as per its dynamics. Now, for this sub-sequence, the controller executes once followed by the continuous evolution of the plant over a time window of $3h$ instead of h. Therefore, for this instance of the execution, we propose to replace the controller designed for a sampling period of h with a different controller gain designed for period of $3h$. In general, for a q-length loop skipping sub-sequence, we use a new controller, that is, a switched compensating controller with a sampling period of qh. The remainder of this section presents the detailed methodology for designing the switched compensating controllers.

4.1 Drop Compensating Switched Multi-rate Controllers

We define some important terminologies as follows:

Definition 4.1 (Loop Skipping Sub-sequence). A loop skipping sub-sequence of an l-length recurrent loop execution pattern $s[1 \dots l]$ is a q-length sub-string of s having the form 10^{q-1}. In other words, the sub-sequence starts with 1 and is followed by $(q-1)$ number of 0s. Since the loop execution pattern is recurrent (cyclic), this means that $s[i] = 1$ for some i, $1 \le i \le l$ and the subsequent $(q-1)$ number of 0s are at $s[(i+1)\%l], \dots, s[(i+q-1)\%l]$, respectively, for some $q \in \{1, \dots, l\}$.

For example, the loop execution pattern $s = 0110101110$ has the loop-skipping sub-sequence, 10, at the positions $(s[3], s[4])$ and $(s[5], s[6])$. It has the loop-skipping sub-sequence 100 with the leading 1 at the position $s[9]$, and the two successive 0s at $s[10]$ and $s[1]$. For theoretical convenience, we also have loop-skipping sub-sequences of unit length, namely a 1 followed by no 0s. Such sub-sequences, of the form 1, appear at the positions $s[2]$, $s[7]$, and $s[8]$.

In order to compensate for the drops in a loop-skipping sub-sequence, we propose to replace sub-sequences of the form $(A_1)(A_0)^{q-1}$ with sub-sequences of the form $(A_q)(A_0)^{q-1}$, where A_q represents a suitably modified control law executed in the same computational slot as the A_1 at the head of the sub-sequence. This results in a *switched loop execution pattern* denoted by $q0^{q-1}$ which is formally defined as follows:

Definition 4.2 (Switched Loop Execution Pattern). Given a loop execution pattern $s[1,\ldots,l]$ over $\Sigma = \{0,1\}$, the switched loop execution pattern $\hat{s}[1,\ldots,l]$ defined over $\Sigma = \{0,1,2,\ldots,q\}$ is obtained by replacing each loop skipping sub-sequence 10^{q-1} of s with the sub-sequence $q0^{q-1}$.

For example, the loop execution pattern, $s = 1110101100$, is replaced by the *switched loop execution pattern*, $\hat{s} = 1120201300$, where each occurrence of loop-skipping sub-sequences 10 and 100 of s are replaced by 20 and 300, respectively. We refer to the controller, A_1, as the *base controller*, and its sampling period as the *base sampling period*. There are many ways in which the compensating controller, A_q, can be designed for the sub-sequence $(A_q)(A_0)^{q-1}$. In this article, we use standard control design techniques for preparing a controller with sampling period $q \times h$ to define A_q, where h is the sampling period of the uniform controller A_1. In other words, A_1 is designed assuming that the controller will execute again after a period of h, where as, A_q is designed with the knowledge that no control execution will take place within the next $q \times h$ units of time.

It is important to note that, replacing 10^{q-1} with $q0^{q-1}$ merely replaces the control law at the same execution slot, without affecting the runtime of the control task in the ECU. Formally, for a switched loop execution pattern, $\hat{s}[1,2,\ldots,l] = q_1 0^{q_1-1} \cdots q_M 0^{q_M-1}$, of length $l = q_1 + q_2 + \cdots + q_M$, the switched dynamical system evolves as:

$$\begin{aligned} x[l] &= (A_0)^{q_M-1} A_{q_M} \cdots (A_0)^{q_j-1} A_{q_j} \ldots (A_0)^{q_1-1} A_{q_1} x[0] \\ &= A_{\hat{s}[l]} \cdots A_{\hat{s}[j]} \cdots A_{\hat{s}[1]} x[0] \end{aligned} \tag{4}$$

In the above, for convenience of notation, we use $A_{\hat{s}[j]}$ to denote the product matrix for $(A_0)^{q_j-1}A_{q_j}$. Also, we define the set of *pattern associated sampling periods* for a switched loop execution pattern, $\hat{s}[1,2,\ldots,l]$ as the set $\{q_1 \times h, \ldots, q_m \times h\}$ corresponding to the m loop skipping sub-sequences of distinct lengths. For example, corresponding to the switched loop execution pattern, $\hat{s} = 112020113001201$, the pattern associated sampling periods are $\{h, 2h, 3h\}$ since $\hat{s}$ has three distinct loop skipping sub-sequences, namely, 1, 20, and 300.

Stable Controller Design during Switching. For a switched loop execution pattern of a control loop, we use linear quadratic regulator (LQR)-based techniques to design the m optimal controllers for each of the m distinct pattern associated sampling periods. In the LQR method, given the plant model and the sampling period, the objective is to find the control input, u, that minimizes the quadratic cost function J, subject to the linear system dynamics [4] with

$$J = \lim_{\tau \to \infty} \int_0^{\tau} (x^{\mathsf{T}}(t)Qx(t) + u^{\mathsf{T}}(t)Ru(t))\, dt.$$

In J, $Q \succcurlyeq 0$ and $R \succ 0$ are symmetric weighting matrices representing the relative importance that the control designer gives to the state trajectories and control effort respectively.

In a switched dynamical system, it may happen that, in spite of having stable controller for all individual state (corresponding to a choice of sampling period), there might exist some switching pattern for which the entire switched system becomes unstable [29, 42]. To overcome such situation, one possible research direction is to ensure stability of switched systems under *arbitrary switching* (see [29] for details). The necessary condition for stability under arbitrary switching, is finding the existence of common quadratic Lyapunov function (CQLF) for all its states. The existence of CQLF can be expressed as linear matrix inequalities (LMIs) [7] and there exists several well-defined procedures for this [29]. Among these existing procedures, we follow the one provided in [42] to design the optimal controllers for different sampling periods, so that the controllers are robustly stable against any arbitrary switching between themselves.

More specifically, if for a control loop, there exists d number of different sampling periods, namely, $\{h, 2h, \ldots, dh\}$, with which the system is stabilizable, then following the LMI-based approach of [42] (see Section 3 of [42] for details), we design the optimal stable controller for each sampling period $jh, j \in \{1, 2, \ldots, d\}$ and compute the respective closed loop dynamic matrix A_d. Now, for a given switched loop execution pattern of that loop, the set of m optimal stable controllers for each of the m distinct pattern associated sampling periods, are the subset of this set of d optimal stable controllers of that loop, and the dynamic matrices, $A_1, \ldots, A_m \in \{A_j, j = 1, 2, \ldots, d\}$, designed in this way then become the choices of dynamic matrices as discussed earlier in Equation (4).

The design of stable controllers in this way basically guarantees closed loop stability of the entire system when we allow slot exchanges in the schedule of the control loops following a switched loop execution pattern and obtain a new switched loop execution pattern (detailed description of slot exchanging method is given in Section 5).

4.2 Cost of Switched Loop Execution Patterns

The quadratic cost function provides a measure of the control performance, i.e., the quality of control. Following the notion of quadratic cost functions defined for a switched system [45], we define the quadratic cost function for the switched controller of Equation (4) over a finite sampling window $[0, l-1]$ as follows:

$$\begin{aligned} J &= \max_{x[0]\in\mathbb{R}^n} \sum_{k=0}^{l-1} x^{\mathsf{T}}[k]Qx[k] \qquad (5)\\ &= \max_{x[0]\in\mathbb{R}^n} \sum_{k=0}^{l-1} (A_{\hat{s}[k]} \cdots A_{\hat{s}[1]}x[0])^{\mathsf{T}} Q (A_{\hat{s}[k]} \cdots A_{\hat{s}[1]}x[0]), \end{aligned}$$

where Q is a positive definite quadratic weight matrix representing the extent of the deviation of state $x[k]$ from the set point. The right-hand side of Equation (5) can be rewritten as:

$$\begin{aligned} &\max_{x[0]\in\mathbb{R}^n} x[0]^{\mathsf{T}} \left(\sum_{k=0}^{l-1} (A_{\hat{s}[k]} \cdots A_{\hat{s}[1]})^{\mathsf{T}} Q (A_{\hat{s}[k]} \cdots A_{\hat{s}[1]}) \right) x[0] \\ &= \max_{x[0]\in\mathbb{R}^n} x[0]^{\mathsf{T}} \mathcal{T}_{\hat{s}}\, x[0], \end{aligned}$$

where $\mathcal{T}_{\hat{s}} = \sum_{k=0}^{l-1} (A_{\hat{s}[k]} \cdots A_{\hat{s}[1]})^{\mathsf{T}} Q (A_{\hat{s}[k]} \cdots A_{\hat{s}[1]})$ is a positive definite matrix and the quadratic form $x[0]^{\mathsf{T}}\mathcal{T}_{\hat{s}}x[0]$ is bounded as follows:

$$\lambda_{min}(\mathcal{T}_{\hat{s}})\, ||x[0]||^2 \leq x[0]^{\mathsf{T}}\mathcal{T}_{\hat{s}}x[0] \leq \lambda_{max}(\mathcal{T}_{\hat{s}})\, ||x[0]||^2, \qquad (6)$$

where $\lambda_{min}(\mathcal{T}_{\hat{s}})$ and $\lambda_{max}(\mathcal{T}_{\hat{s}})$ are the minimum and maximum eigenvalues of $\mathcal{T}_{\hat{s}}$. Subject to the constraint $||x|| \leq \chi$, where χ is a constant, we have $J = \lambda_{max}(\mathcal{T}_{\hat{s}}) \cdot \chi^2$ since $x[0] \in \{x \mid ||x|| \leq \chi\}$ (as shown in [28]). Given two patterns with control costs J_1 and J_2 , respectively, it may be observed that the term χ^2 is the same in both J_1 and J_2, and therefore, the cost comparison may be done only with respect to the maximum eigenvalue. Hence, we define a quadratic cost function for a switched loop execution pattern, $\hat{s}$, as follows:

Let $\oplus(\hat{s}, i)$ represent the i-place cyclic right shift of $\hat{s}[1, \ldots, l]$. We define the quadratic cost of the pattern $\hat{s}$ as the maximum among the maximum eigenvalues for $\mathcal{T}_{\oplus(\hat{s},i)}$, $1 \leq i \leq l$. Since the pattern recurs after every l steps, the different possible cyclic shifts capture all the different l-length evolution possible for the system. Overall, for an l-length switched loop execution pattern $\hat{s}$, we have,

$$Cost(\hat{s}[1 \cdots l]) = max\{\lambda_{max}(\mathcal{T}_{\oplus(\hat{s},i)}) \mid 1 \leq i \leq l\}. \tag{7}$$

Let $\bar{\hat{s}} \triangleq \{\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_n\}$ represent a schedulable combination of switched loop execution patterns of n control loops. We define the combined cost of the loops in $\bar{\hat{s}}$ as $Cost(\bar{\hat{s}}) \triangleq \sum_{\hat{s}_i \in \bar{\hat{s}}} Cost(\hat{s}_i)$. In a more general setting, the combined cost of the loops may also be defined in other ways, for example, as a weighted sum of the costs of the individual loops.

Lower the combined cost, better is the control performance, since any high value of J for any loop indicates either a high deviation of the desired state or a high control effort needed to bring the state to its set point.

5 EXCHANGING SLOTS IN STATIC SCHEDULES

The methodology outlined in Section 4 aims to improve the control cost of individual control loop execution patterns by developing corresponding switched loop execution patterns involving multiple control laws. This section focuses on improving the combined cost of the set of control loops sharing an ECU by exchanging computational slots between control loops.

Let $\bar{s} \triangleq \{s_1, s_2, \ldots, s_n\}$ represent a schedulable combination of switched loop execution patterns of n control loops. Each pattern in $\bar{s}$ is recurrent, but the lengths of the patterns as well as their base sampling periods may differ. Let, $h_1, h_2, \ldots, h_n$ be the base sampling periods of the n control loops, and $l_1, l_2, \ldots, l_n$ be the lengths of the loop execution patterns $s_1, s_2, \ldots, s_n$, respectively. The period of recurrence of the set of patterns taken together is therefore given by $t_B = lcm(l_1 \times h_1, l_2 \times h_2, \ldots, l_n \times h_n)$. We use $\rho_i[1 \cdots L_i]$ to denote the extension of the loop execution pattern $s_i[1 \cdots l_i]$ up to the combined period t_B, where $L_i = \frac{t_B}{h_i}$. The qth bit of s_i appears as the $(q + \eta l_i)$-th bit in ρ_i for $\eta = 0, 1, \ldots, \frac{L_i}{l_i}$.

Given a static schedule of two control loops running on an ECU, we explore the option of exchanging a loop execution slot of one loop with a loop-skipping slot (or drop) of the other. In our representation, this is expressed by flipping a 1 to 0 in the extended loop execution pattern of the former and flipping a 0 to 1 in the extended loop execution pattern of the latter.

Example 5.1. Suppose $s_1 = 1011011$ and $s_2 = 0100101$ are the recurrent loop-execution patterns for two control loops with sampling periods 2 ms and 3 ms, respectively. As both the loop-execution patterns are of length 7, the combined period of recurrence is $lcm(7 \times 2, 7 \times 3,)$ms = 42 ms. The patterns extended up to 42 ms are, respectively, $\rho_1 = 101101110110111011011$ and $\rho_2 = 01001010100101$, and the corresponding switched-loop execution patterns are $\hat{\rho}_1 = 201201120120112012011$ and $\hat{\rho}_2 = 03002020300202$. The control loops have execution time as 1ms and 1.5 ms, respectively. Our intention is to explore the trading of computation slots between the control loops and to study the effect of such exchanges on the combined control performance. Figure 6 demonstrates the notion of a *computational slot exchange* in the first 12ms of the schedule on a shared ECU. If the first loop in Figure 6 skips its task (green-colored) scheduled at the slot starting at 10ms, then the second loop can execute its task (red-colored) in that computational slot. This means that the loop execution pattern of the first loop is now $\rho_1' = 10110\mathbf{0}110110111011011$, which is obtained by replacing the 6th bit of ρ_1 by a 0, and the second loop execution pattern is now $\rho_2' = 010\mathbf{1}1010100101$, which is obtained by replacing the 4th bit of ρ_2 by a 1. This also changes the switched loop execution patterns to $\hat{\rho}_1' = 201300120120112012011$ and $\rho_2' = 02012020300202$, respectively.

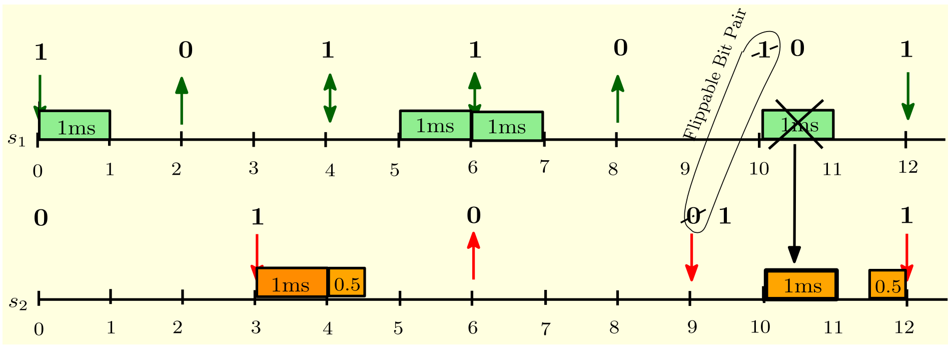


Fig. 6. The notion of slot exchange.

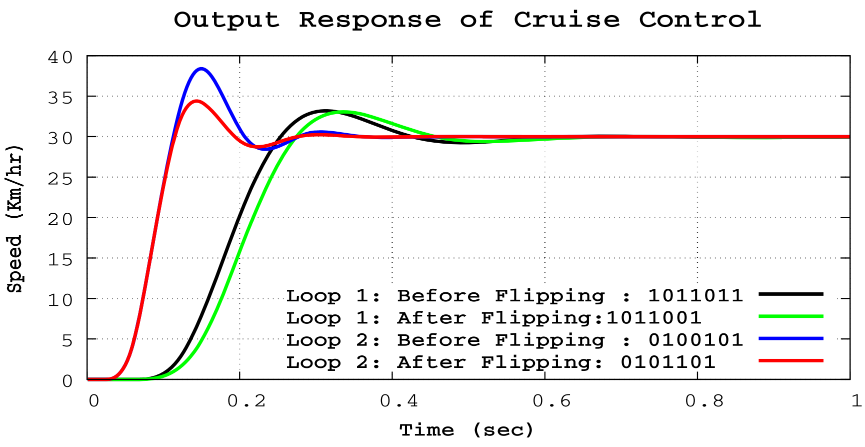


Fig. 7. Performance improvement due to slot exchange.

Such trading of computational slots can be leveraged to improve the combined control cost. For example, the combined control cost before the exchange indicated above is:

$$Cost(\hat{\rho}_1) + Cost(\hat{\rho}_2) = 1582.4 + 19055 = 20637.4.$$

After the exchange, the combined control cost is:

$$Cost(\hat{\rho}_1') + Cost(\hat{\rho}_2') = 5630.6 + 8797.2 = 14427.8.$$

In this case, the control cost reduces significantly. The effect of this slot exchange on closed loop control performance can be observed from the step responses of these two loops. Figure 7 shows the step response curves before and after the flipping of the bits. We observe a significant improvement in the response of the second loop (the red and blue curves), which comes at the expense of minor degradation in the response of the first loop (the black and green curves).

Post-processing of recurrent switched loop execution patterns, as illustrated in Example 5.1, has not been exploited in existing literature on embedded control. A significant contribution of this article is to develop the first formal exposition of the problem and present the preliminary algorithms for this task.

5.1 Conditions for Slot Exchange

The design of the controller A_q for a loop-skipping sub-sequence 10^{q-1}, as described in Section 4, is possible only if there exists a stable controller with sampling period of $q \times h$, where h is the sampling period of the base controller, A_1. In general, there will exist a maximum value of q, such that a stable controller exists with a sampling period $q \times h$, but no stable controller exists with a sampling period of $(q+1) \times h$. Note that, as discussed in Section 4.1, we design the stable controllers for multiple sampling periods using the LMI-based approach proposed in [42] so that stability remains guaranteed even during any arbitrary switching among the controllers. When flipping a bit in a loop execution pattern, we must guarantee that some stable controller exists for the affected loop skipping sub-sequence. Flipping a 0-bit to a 1-bit is always stability preserving, but for flipping a 1-bit to a 0-bit needs our attention.

Definition 5.2 (Stability Preserving Bit Flip). Flipping a 1-bit to a 0-bit in an L-length extended loop execution pattern, ρ, is stability preserving if and only if stable controllers exist for each of the pattern associated sampling periods of the modified switched loop execution pattern, $\hat{\rho}'$, where ρ' is obtained due to the flipping of 1 bit in ρ.

Definition 5.3 (Stable Complementary Bit Pair). Given the extended patterns ρ_i and ρ_j for the ith and jth control loops, the bit pair $\langle \rho_i[q], \rho_j[r] \rangle$ is said to be a stable complementary bit pair if and only if $\rho_i[q] = 1$, $\rho_j[r] = 0$, and the flips of $\rho_i[q]$ and $\rho_j[r]$ are stability preserving.

A stable complementary bit pair, $\langle \rho_i[q], \rho_j[r] \rangle$, is eligible for an exchange of computational slots only when the execution slots, $[t, t+h_i)$ and $[t', t'+h_j)$, corresponding to $\rho_i[q]$ and $\rho_j[r]$, overlap in time and the execution time relinquished by flipping $\rho_i[q]$, creates the gap for fitting the control task of the other loop.

Definition 5.4 (Flippable Bit Pair). A stable complementary bit pair $\langle \rho_i[q], \rho_j[r] \rangle$ in two extended patterns, ρ_i and ρ_j, is a flippable bit pair if the following conditions hold:

(1) The execution slot, $[t, t+h_i)$ for $\rho_i[q]$ overlaps with the slot $[t', t'+h_j)$ corresponding to the drop at $\rho_j[r]$.
(2) Within the time window, $[t', t'+h_j)$, the relinquished bandwidth, namely $[t, t+h_i) \cap [t', t'+h_j)$, when added with the existing free time, exceeds the worst case execution time of the second loop, thereby enabling $\rho_j[r]$ to convert from a drop instance to an execution instance.

In our algorithm presented later, we use a Boolean function $FLIP(\rho_i[q], \rho_j[r])$ which is true when $\langle \rho_i[q], \rho_j[r] \rangle$ is a flippable bit pair, and false otherwise. Example 5.5 illustrates these facts as follows:

Example 5.5. In Figure 8, consider the stable complementary bit pair $\langle \rho_2[4], \rho_1[5] \rangle$. The time windows corresponding to $\rho_2[4]$ and $\rho_1[5]$ are $[9, 12)$ and $[8, 10)$, respectively. Though these intervals overlap, $\langle \rho_2[4], \rho_1[5] \rangle$ is not a flippable bit pair (hence marked with a cross in Figure 8). This is because:

(1) We need 1ms of time in the window $[8, 10)$ to convert $\rho_1[5]$ into an execution slot.
(2) The existing free time in the window $[8, 10)$ is 0.
(3) The time gained within the window $[8, 10)$ by dropping $\rho_2[4]$ is 0.5ms. This time, when added with the existing free time in the window $[8, 10)$ is not sufficient to execute the 1 ms task.

On the other hand, $\langle \rho_2[4], \rho_1[6] \rangle$ is a flippable bit pair (hence marked with a tick in Figure 8) because time intervals $[10, 12)$ (for $\rho_1[6]$) and $[9, 12)$ (for $\rho_2[4]$) overlap, and the total free time

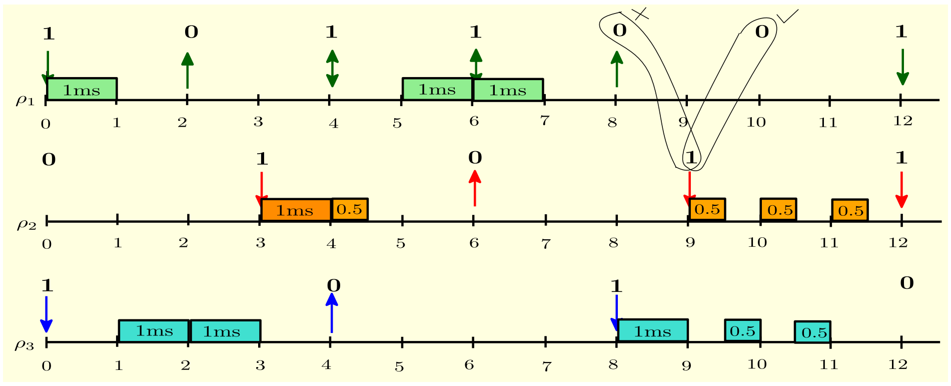


Fig. 8. Bit flipping condition.

within the window $[10, 12)$ after dropping $\rho_2[4]$ is 1.5 ms, namely 0.5 ms of existing free time plus 1 ms of time gained by dropping $\rho_2[4]$.

Next, we formally define the notion of gain in control cost obtained due to a slot exchange between two patterns as specified by the respective flippable bit pair.

Definition 5.6 (Gain from Flippable Bit Pair). The gain, $\mathcal{G}(\rho_i[q], \rho_j[r])$, from a flippable bit pair, $\langle\rho_i[q], \rho_j[r]\rangle$, in two extended loop execution patterns, ρ_i and ρ_j is:

$$[Cost(\hat{\rho}_i) + Cost(\hat{\rho}_j)] - [Cost(\hat{\rho}'_i) + Cost(\hat{\rho}'_j)],$$

where ρ'_i and ρ'_j are the extended loop execution patterns obtained after the exchange of the execution slots, and $\hat{\rho}'_i$ and $\hat{\rho}'_j$ are the corresponding switched loop execution patterns of ρ'_i and ρ'_j, respectively.

Note that the gain of a flippable bit pair is the difference between the combined control cost of the respective switched loop execution patterns obtained before and after the bit flips.

5.2 Algorithmic Framework

In our methodology, we consider the flippable bit pairs with positive gain and progressively reduce the control cost. Given a set of recurrent loop execution patterns for a set of control loops, and a schedule of execution of the control tasks, we find all the flippable bit pairs and their gains. It is possible that multiple flippable bit pairs with positive gain are available between any two loop execution patterns, but when both the pairs are used for exchanging slots, the combined cost becomes worse. This is demonstrated by the following example:

Example 5.7. Consider the pair of patterns ρ_1 and ρ_2 of Example 5.1. Recall that the combined cost of these patterns was 20637.4. The flippable bit pair, $\langle\rho_1[4], \rho_2[3]\rangle$, can be used to replace ρ_1 and ρ_2 with $\rho''_1 = 10100111011011101101 1$ and $\rho''_2 = 01101010100101$, respectively. The corresponding switched loop execution patterns are $\hat{\rho}''_1 = 2030011201201120 12011$ and $\hat{\rho}''_2 = 01202020300202$, respectively. We find that:

$$Cost(\hat{\rho}''_1) + Cost(\hat{\rho}''_2) = 5719.1 + 8797.2 = 14516.3$$

and therefore:

$$\mathcal{G}(\rho_1[4], \rho_2[3]) = 20637.4 - 14516.3 = 6121.1.$$

In Example 5.1, we had shown that:

$$\mathcal{G}(\rho_1[6], \rho_2[4]) = 20637.4 - 14427.8 = 6209.6.$$

If we use both the flippable bit pairs, $\langle \rho_1[4], \rho_2[3] \rangle$ and $\langle \rho_1[6], \rho_2[4] \rangle$, at the same time, then we get $\rho_1''' = 10100011011011101101 1$ and $\rho_2''' = 01111010100101$, with the corresponding switched loop execution patterns of $\hat{\rho}_1''' = 20400012012011201201 1$ and $\hat{\rho}_2''' = 01112020300102$. The combined quadratic cost in this case is:

$$Cost(\hat{\rho}_1''') + Cost(\hat{\rho}_2''') = 13448 + 5560.2 = 19008.2$$

and the gain is $20637.4 - 19008.2 = 1629.2$. This gain is inferior to each of the individual gains from $\langle \rho_1[4], \rho_2[3] \rangle$ and $\langle \rho_1[6], \rho_2[4] \rangle$.

Example 5.7 shows that gains from multiple flippable bit pairs between the same pair of loops do not add up. On the other hand, gains from flippable bit pairs involving disjoint loops add up. Therefore, for algorithmic convenience, we define our optimization goal in terms of flippable bit pairs involving disjoint loops. The problem statement thus becomes:

Given the set of all flippable bit pairs among a set of recurrent loop execution patterns, we wish to find the combination of flippable bit pairs with maximum total gain, under the constraint that the chosen flippable bit pairs must involve disjoint pairs of loops.

In other words, the problem can be stated as:

For n control loops, given a static schedule, $\mathbb{S}$, of n extended loop execution patterns, $\bar{\rho} = \{\rho_1, \ldots, \rho_n\}$, we wish to find a refined static schedule, $\mathbb{S}'$, of n modified patterns, $\bar{\rho'} = \{\rho_1', \ldots, \rho_n'\}$, obtained by flipping the bits of the patterns in $\bar{\rho}$, so that, flippable bit pairs must involve disjoint pairs of loops and

$$\sum_{\hat{\rho}_i \in \bar{\hat{\rho}}} Cost(\hat{\rho}_i) - \sum_{\hat{\rho}_i' \in \bar{\hat{\rho}}'} Cost(\hat{\rho}_i') \text{ is maximized,}$$

where $\hat{\rho}_i$ and $\hat{\rho}_i'$ are the switched loop execution patterns corresponding to ρ_i and ρ_i', respectively.

Solution to the Optimization Problem. Our problem directly maps into the *maximum weight independent subset problem* (MWIS) [41]. We define a graph in which the nodes represent flippable bit pairs. The gain from a flippable bit pair defines the weight of the corresponding node. Nodes representing flippable bit pairs with one or more common loops are adjacent. In other words, independent sets in the graph represent combinations of flippable bit pairs involving disjoint pairs of loops, and our goal is to find the combination having maximum gain, and therefore the independent set having maximum weight. Algorithm 1 prepares the graph by leveraging the theory presented in this article. The MWIS problem on the graph is then solved using standard *Integer Linear Programming* (ILP) techniques.

The solution to the MWIS problem identifies the chosen combination of flippable bit pairs and then we apply the corresponding slot exchanges to create the final schedule. Next, we describe Algorithm 1 in detail. Lines 1–4 of Algorithm 1 finds all stability preserving bit flips and marks the respective bit in each extended loop execution pattern, $\rho_i \in \bar{\rho}$. Lines 5–9 searches for all possible flippable bit pairs among the n patterns. For a selected bit pair in line 6, whether it is a stable complementary bit pair or not, is checked in line 7. If it is so, then the flipping condition (see Definition 5.4) is checked in Line 8 to decide if it is a flippable bit pair or not. The nodes and edges of the graph are defined in Lines 10–12, while the node weights are assigned in Lines 13–14. Line 15 returns the graph.

Example 5.8. Let us illustrate the graph construction with the previous example given in Example 5.5. Assume that, the set of stabilizable sampling periods for the loops are $H_1 = \{h_1,$

ALGORITHM 1: Graph_Construction for MWIS

```
Input: The Static Schedule: $\mathbb{S}$, Pattern Lengths: $\{L_1, L_2, \ldots, L_n\}$, Initial Extended Patterns:
       $\bar{\rho} = \{\rho_1, \rho_2, \ldots, \rho_n\}$, Initial Cost: $Cost(\hat{\rho}_i), \forall i = 1, 2, \ldots, n$
Output: Graph $G(V, E)$ corresponding to $\mathbb{S}$ and $\bar{\rho}$
// Identify Stability Preserving Bit Flips
1  for each $\rho_i \in \bar{\rho}$ do
2      for each bit $\rho_i[q]$ of the pattern $\rho_i, q = 1, 2, \ldots, L_i$ do
3          if flipping of $\rho_i[q]$ is stability preserving then
4              Mark the bit $\rho_i[q]$ ;                                  // Stable Flip
// Find flippable bit pairs
5  for each pair of patterns, $\rho_i, \rho_j \in \bar{\rho}$ such that $i \neq j$ do
6      for each bit pair $(\rho_i[q], \rho_j[r])$, $i = 1, \ldots, L_i$, $j = 1, \ldots, L_j$ do
7          if both the bits $\rho_i[q]$, $\rho_j[r]$ are marked && $\rho_i[q] == 1$ && $\rho_j[r] == 0$ then
8              if $FLIP(\rho_i[q], \rho_j[r])$ is true then
9                  Set the pair $(\rho_i[q], \rho_j[r])$ as flippable;
// Construct the Graph $G(V, E)$
10 for each flippable bit pair $(\rho_i[q], \rho_j[r])$ do
11     Add a node $\langle\rho_i[q], \rho_j[r]\rangle$ in $V$ of $G(V, E)$;
// Give an edge between two nodes if they share same pattern id
12 Put an edge between nodes $\langle\rho_i[q], \rho_j[r]\rangle$ and $\langle\rho_v[q'], \rho_w[r']\rangle$, on satisfaction of the condition:
   $(i == v \vee i == w) \vee (j == v \vee j == w)$;
13 for each node $\langle\rho_i[q], \rho_j[r]\rangle$ do
       // Assign weight of a node as the gain of that respective bit pair
14     Assign the weight: $\mathtt{W}(\langle\rho_i[q], \rho_j[r]\rangle) = \mathcal{G}(\rho_i[q], \rho_j[r])$;
15 return $G(V, E)$;
```

$2h_1, 3h_1\}$, $H_2 = \{h_2, 2h_2, 3h_2\}$ and $H_3 = \{h_3, 2h_3\}$, respectively, with $h_1 = 2$ ms, $h_2 = 3$ ms, and $h_3 = 4$ ms. Figure 9 shows the partial construction of the graph obtained from the given static schedule of Example 5.5, with respect to the flippable bit pair $(\rho_2[4], \rho_1[6])$ (encircled in the figure). The bits marked as cross, i.e., $\rho_1[7], \rho_2[2], \rho_2[7], \rho_3[3], \rho_3[6]$, do not ensure stability preserving bit flip and that is why no node has been defined including those bits. For other bits of all the loop execution patterns, nodes are defined for all the flippable bit pairs. For the sake of clarity, in Figure 9, nodes corresponding to the flippable bit pairs $(\rho_3[1], \rho_2[1])$, $(\rho_2[4], \rho_1[6])$, and $(\rho_3[7], \rho_1[6])$ are specifically shown. In this partially constructed graph, the upper portion of the label of a node indicates the respective bit positions of loop execution patterns, where the bit position with the bit set as '1' comes first. For example, for the node $\langle\rho_2[4], \rho_1[6]\rangle$, the upper portion of the label is '4,6'. The lower portion of the label indicates the loop ids for the '1' bit and '0' bit, respectively, e.g., for the node $\langle\rho_2[4], \rho_1[6]\rangle$, the lower portion of the label is '2,1'. Any other node defined using ρ_1 and ρ_2 has an edge from the node $\langle\rho_2[4], \rho_1[6]\rangle$.

Complexity of Algorithm 1. The complexity of the graph construction algorithm is as follows: Let, $L_{max} \triangleq max\{L_1, \ldots, L_n\}$. The complexity of Lines 1–4 is upper bounded by $O(nL_{max})$. Finding all flippable bit pairs in Lines 5–9 is done in $O(n^2 L_{max}^2)$ time, because we consider each of the n^2 pairs of loops and within each pair we examine $O(L_{max}^2)$ bit pairs. The number of nodes in the graph is $O(n^2 L_{max}^2)$, and the number of edges is quadratic in that number. Though the

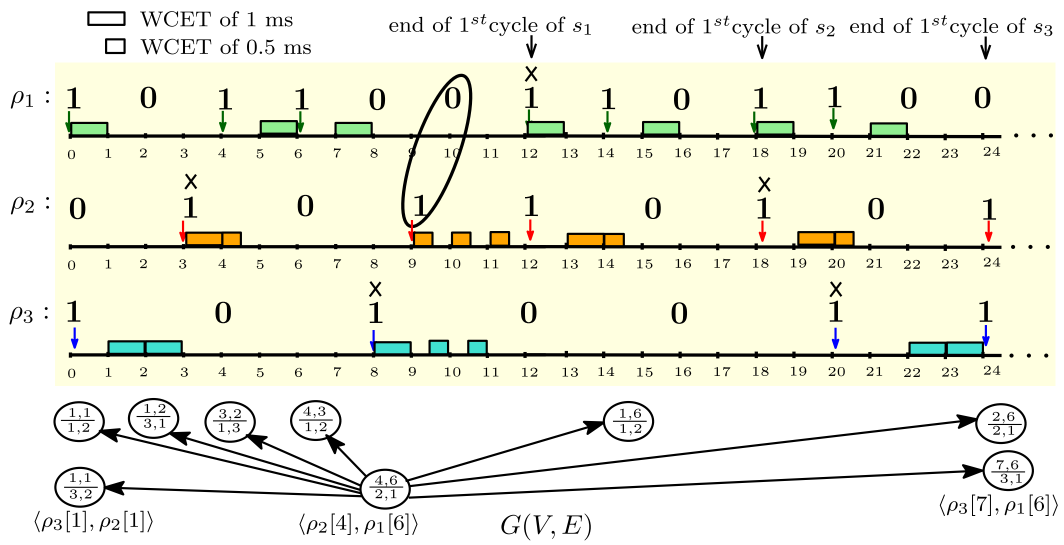

WCET of 1 ms
WCET of 0.5 ms
end of 1st cycle of s_1
end of 1st cycle of s_2
end of 1st cycle of s_3
ρ_1 :
ρ_2 :
ρ_3 :
$\langle\rho_3[1], \rho_2[1]\rangle$
$\langle\rho_2[4], \rho_1[6]\rangle$
$\langle\rho_3[7], \rho_1[6]\rangle$
$G(V, E)$

Fig. 9. Partially constructed graph from the given static schedule.

construction of the graph works in time polynomial in its input size, the overall methodology has higher complexity. This is because the input to Algorithm 1 is the set of patterns extended up to the LCM of their lengths, and is therefore exponentially larger than the size of the recurrent loop execution patterns. Moreover, no polynomial-time algorithm is known for the MWIS problem (which is NP-hard) [17], and our ILP solution does not guarantee polynomial-time solution. Nevertheless, since the number of control loops sharing an ECU and the lengths of the recurrent loop execution patterns are typically not too large, we do not have any serious complexity concerns in applying the methodology.

A Note on Optimality. It is important to note that our approach does not guarantee to produce scheduling patterns with the minimum combined control cost. The generic problem of finding a cost-optimal schedule (where cost refers to combined control cost) for multiple control loops, each switching between multiple control laws to compensate for the drops, exhibits very high state space requirements and thereby very high complexity. We do not find an optimal solution to this problem. Since this generic problem is so hard, existing (sub-optimal) solutions leave room for further improvement through local optimization, which is the goal of our approach. The problem that we solve is one of local optimization where we locally exchange computation slots and switch between control laws to improve the control performance. This, too, is a problem of significant complexity considering the number of ways in which slot exchanges can be done. We therefore further reduce the scope of the problem by defining flippable bit pairs and casting the problem as stated earlier in Section 5.2. Therefore, in summary, our approach produces an optimal solution for the targeted local optimization problem, but the problem itself is a part of a bigger and more complex optimization problem, for which no feasible optimal approach exists in the literature.

A Note on Overhead. Here we discuss the design and runtime overheads associated with our proposed method. The overheads in the control and schedule design steps precede the actual deployment of the controller. Since the proposed method is an offline step, it is not constrained by the computational limitations of the embedded platform. The task of finding an optimal combined schedule with multiple controllers and drops is a computationally hard problem considering the size of the design space. We are therefore proposing a local optimization approach, where we deal

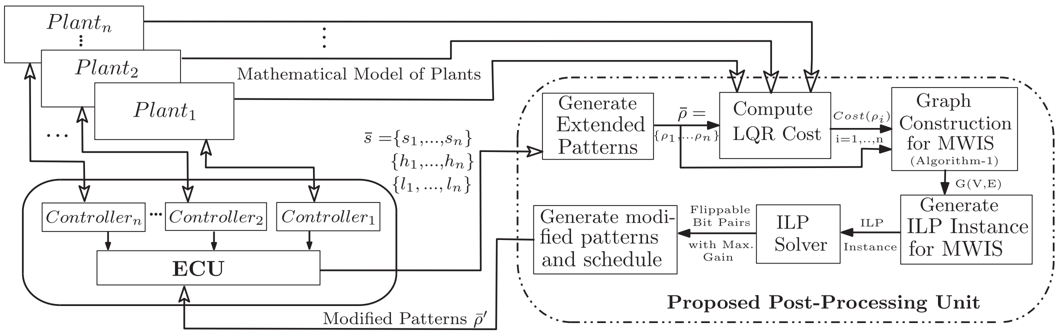

$Plant_n$
$Plant_2$
$Plant_1$
Mathematical Model of Plants
$\bar{s} = \{s_1,...,s_n\}$
$\{h_1,...,h_n\}$
$\{l_1,...,l_n\}$
$Controller_n$
$Controller_2$
$Controller_1$
ECU
Modified Patterns $\bar{\rho}'$
Generate Extended Patterns
$\bar{\rho} = \{\rho_1...\rho_n\}$
Compute LQR Cost
$Cost(\rho_i)$ i=1,...,n
Graph Construction for MWIS (Algorithm-1)
G(V,E)
Generate ILP Instance for MWIS
ILP Instance
ILP Solver
Flippable Bit Pairs with Max. Gain
Generate modified patterns and schedule
Proposed Post-Processing Unit

Fig. 10. The overall methodology and the automated tool-flow.

with local slot exchanges and local choices of alternative controllers. Moreover, we use recurrent patterns of limited length. These choices limit the complexity of our approach, and even though we map to the MWIS problem, we obtain solutions in time that are reasonable for this offline design step. The runtime overhead is primarily of two types, namely the memory overhead for accommodating multiple control laws, and the time overhead for dynamically choosing the control law following the execution pattern. Typically, the variants of the control law that are suitable for various sampling periods differ in some of the parameters (coefficients/matrix entries) of the control function, which means that the control code does not actually multiply. Moreover, the number of variants are either not very high naturally, or may be limited as a design choice for our methodology. The runtime switching overhead is not very significant, and that is why a reasonably large body of literature is actively considering controllers with various (m,k)-firm schedules of execution.

The Automated CAD Tool Support. We have implemented an automated CAD framework which considers as input the scheduling details of a shared architectural platform as shown in Figure 10. We consider the static schedule of the control loops that are created by an embedded system designer based on his knowledge about control design and platform. Note that it is possible to miss this optimized solution during the design of the controller and platform parameters since traditional design practice relies on the separate design and optimization of control and platform parameters. Hence, the post-processing solution that our automated framework potentially provides may have been missed in the human-driven design cycle of the control and scheduling solution. This is where our proposed CAD framework performs post-processing optimizations.

The tool-flow automates the synthesis and implementation process of the proposed post-processing methodology. The component-wise detailed description of the tool-flow is highlighted in the post-processing unit of Figure 10. The inputs to the tool are the corresponding loop execution patterns ($\{s_1, \ldots, s_n\}$), sampling periods ($\{h_1, \ldots, h_n\}$), and the pattern lengths ($\{l_1, \ldots, l_n\}$) of all the control loops. In the first phase, as discussed earlier, the extended pattern ρ_i is generated from s_i using Equations (8) and (9):

$$t_B = lcm(l_1 \times h_1, l_2 \times h_2, \ldots, l_n \times h_n) \text{ and } L_i = \frac{t_B}{h_i} \tag{8}$$

$$\rho_i[q + \eta l_i] = s_i[q] \; \forall i = 0, 1, \ldots \frac{L_i}{l_i}, \; \forall \eta = 0, 1, \ldots, \frac{L_i}{l_i} \tag{9}$$

Next, in the compute LQR cost block, the control cost, $Cost(\hat{\rho}_i)$, of each switched loop execution pattern $\hat{\rho}_i$ corresponding to ρ_i is computed using Equation (7). Together with the extended loop execution patterns and the initial control cost of the respective switched loop execution patterns, in next phase, the graph is constructed following Algorithm 1. Next, taking the graph $G(V, E)$ as the input the following ILP instance is generated for the MWIS problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{\mathtt{v_i} \in V} \mathtt{W}(\mathtt{v_i})\varkappa(\mathtt{v_i}) \\ \text{subject to} \quad & \varkappa(\mathtt{v_i}) + \varkappa(\mathtt{v_j}) \leq 1 \qquad (\mathtt{v_i}, \mathtt{v_j}) \in E \\ & \varkappa(\mathtt{v_i}) \in \{0, 1\} \qquad \mathtt{v_i} \in V. \end{aligned} \tag{10}$$

Here, the variable $\varkappa(\mathtt{v_i})$ for each node $\mathtt{v_i} \in V$ indicating whether $\mathtt{v_i}$ is chosen in the independent set or not. $\mathtt{W}(\mathtt{v_i})$ is the weight of the node $\mathtt{v_i} \in V$. The ILP instance of the MWIS problem is then forwarded to an ILP solver for generating the solution. In this CAD framework, as the underlying ILP Solver, we have used IBM CPLEX [1], especially, the Python API of CPLEX. The solution generated by the solver gives the set of flippable bit pairs with maximum gain in combined control cost. Next, according to those flippable bit pairs, the slots are exchanged to create the final schedule.

6 EXPERIMENTAL EVALUATION

In the classical approach for design of embedded control, the task scheduling phase is primarily concerned with executing all control tasks within their respective sense-actuation windows, sometimes with drops in loop execution schedules. Our intention is to show that there exists an opportunity for significantly improving the control cost by trading the drops among the multiple control loops executing on a shared ECU. In order to demonstrate this opportunity and the effect of our proposed approach for post-processing the schedules, we use the control loops from four automotive features, namely *cruise control* (CC), *suspension control* (SC), *motor speed* (MS), and *electromagnetic brake* (EB). The dynamics of CC, SC, and MS are adapted from [39], and EB from [12].

The linearized third-order CC system regulates the vehicle speed at a reference level by adjusting the engine throttle angle, namely the control input u. The dynamics matrices are:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.05 & -5.28 & -0.24 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 2.47 \end{bmatrix}, C^{\mathsf{T}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The SC system has four state variables representing the position and velocity of the car, and the position and velocity of the suspension mass. The control input is the force applied to the body by the suspension system. The dynamic matrices for this system are:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -8 & -4 & 8 & 4 \\ 0 & 0 & 0 & 1 \\ 80 & 40 & -160 & -60 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 80 \\ 20 \\ -1120 \end{bmatrix}, C^{\mathsf{T}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The second order MS system controls the rotational speed of the motor by adjusting the motor terminal voltage. The state variables represent the rotational speed and the armature current, respectively. The dynamic matrices are:

$$A = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, C = [1 \;\; 0].$$

The fifth-order EB system has the state variables as motor current, motor angular velocity, motor angular position, caliper velocity, and caliper position. The control input is the applied voltage on

Table 1. Loop Parameters

#	Loop	H (ms)	E (ms)	L	N(H)
1	CC1	40	10	20	3
2	CC2	60	20	20	3
3	CC3	60	15	20	3
4	SC1	20	4	20	3
5	SC2	20	5	20	3
6	SC3	20	10	20	3
7	MS1	80	20	10	3
8	MS2	100	20	10	3
9	EB1	40	8	20	3
10	EB2	60	20	20	3

the motor used to control the braking caliper. The dynamics matrices are adapted from [12].

$$A = \begin{bmatrix} -520 & -220 & 0 & 0 & 0 \\ 220 & -500 & -99999 & 0 & 2\times 10^8 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 66667 & -0.1667 & -1.3333\times 10^7 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, C^{\mathrm{T}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Our experimental setup consists of three CC, three SC, two EB, and two MS plant models whose loop parameters are shown in Table 1. Column 3 shows the base sampling period, H, and Column 6 shows the number of multiples, $N(H)$, of H for which stable controllers exist. Column 4 shows the worst case execution time, E, of the controller. Column 5 shows the length of the recurrent loop execution pattern. For example, Row 4 states that the controller SC1 has a base sampling period of 20ms and task execution time of 4 ms. Further, we are given a recurrent loop execution pattern of length 20 for SC1, and our analysis shows that stable controllers exist for SC1 with sampling periods of 20ms, 40ms, and 60 ms, but no stable controller exists with sampling period of 80 ms and more than that. For all the CC systems, the desired speed limit (i.e., the reference value) is chosen as 30 km/hr. Similarly, for all SC and MS systems, we set the reference values at 0.02 m and 0.5 rad/s, respectively.

We have used the MATLAB-Python interface for our experiments. All the simulations have been performed using MATLAB version R2017b, running on 64-bit Ubuntu 14.04 in a 3.10-GHz Intel Core-i5 machine having 4 GB of memory.

We use the following formula to define the percentage gain in control cost after post-processing the schedules:

$$\%Gain = \frac{Cost_{fin} - Cost_{init}}{Cost_{init}}$$

where $Cost_{fin}$ and $Cost_{init}$ are the cumulative control costs for final and initial loop execution patterns, respectively. We anticipated that the percentage gain will depend on the choice of the initial loop execution patterns of the control loops, and also on the combination of control loops chosen for a shared ECU. We next report these studies separately.

Table 2 reports results on three different choices of the initial loop execution patterns for the feature sets {CC1, CC3, SC2}, {CC1, CC2, SC1, MS1}, and {CC1, CC2, SC3, MS1, MS2}. The combination of loop execution patterns in each row is co-schedulable by design. Column 2 of Table 2 shows the number of flippable bit pairs in these patterns. Column 3 shows that significant improvement

Table 2. *%Gain* versus Loop Execution Pattern

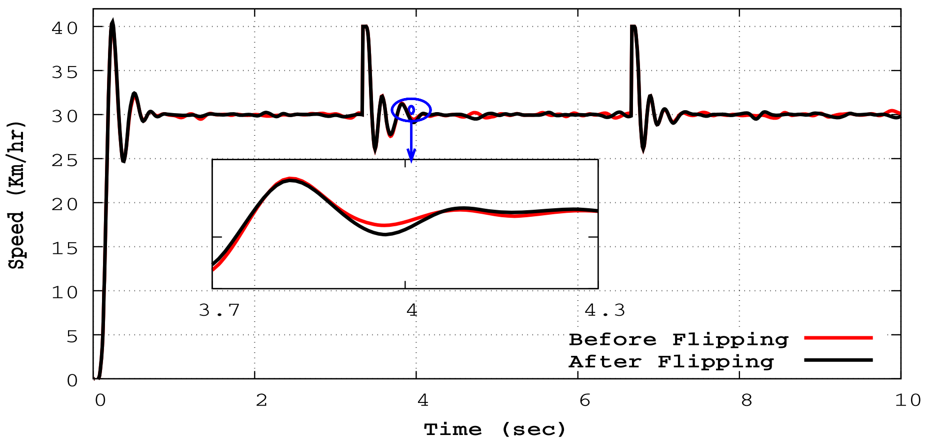
ID	Initial Loop Execution Patterns	FP	*%Gain*
	A. For the set {CC1, CC3, SC2}		
1.	$\{1^4(10)^3 1^4(10)^3,\ 1^4(10)^3 1^2(10)^2 1^4,\ (10)^6 1^8\}$	128	21.35
2.	$\{1^2(100)^2 1^5(100)^2 1,\ (100)1^7(10)^3 1^4,\ 1^5(100)^2 1^9\}$	108	23.56
3.	$\{1^5(100)^2(100)1^6,\ (100)1^7(10)^3 1^4,\ (10)^3 1^8(10)^3\}$	106	22.87
	B. For the set {CC1, CC2, SC1, MS1}		
1.	$\{(100)^2 1^8(10)^3,\ (10)^3 1^{10}(100)1,\ 1^2(100)^2 1^6(100)^2,\ (100)1(10)^2 1^2\}$	789	28.28
2.	$\{(100)^2 1^8(10)^3,\ (10)^3 1^{10}(100)1,\ 1^2(100)^2 1^6(100)^2,\ (100)1^2(10)^2\}$	640	26.95
3.	$\{1^3(10)1^3(10)1^5(10)^2 1,\ (1110)^3 1^2(10)1(100),\ 1^2(10)^2 1^3(10)1^2(10)^2 1^3,\ 1(10)^2 1^2(10)1\}$	1065	23.47
	C. For the set {CC1, CC2, SC3, MS1, MS2}		
1.	$\{(100)^2 1^3(10)1^3(10)^3,\ (10)^3(110)^2 1^2 101001,\ 1^2(100)^2 1^6(100)^2,\ 1001(10)^2 1^2,\ (10)^2 1^2(10)^2\}$	1124	33.23
2.	$\{(10)^3 1^3(10)1^3(10)^3,\ (10)^3(110)^2 1^2 101001,\ 1^2(100)^2 1^6(100)^2,\ 1001(10)^2 1^2,\ 1001^3(10)^2\}$	1176	35.56
3.	$\{(10)^3 1^3(10)1^3(10)^3,\ (10)^3(110)^2 110010 1^2,\ 1^2(100)^2 1^6(100)^2,\ (10)1^4(10)^2,\ 1001(10)^2 1^2\}$	1230	33.95

in control cost (i.e., *%Gain*) can be achieved by using our methodology for finding a suitable combination of the flippable bit pairs, and then using them to improve the control schedule. The results are also indicative of the fact that the gain is not significantly altered by the choice of the initial combination of patterns, given that the initial combinations were of comparable quality in terms of the combined control cost. In other words, our methodology seems to gain regardless of the initial choice of loop execution patterns from among the good ones.

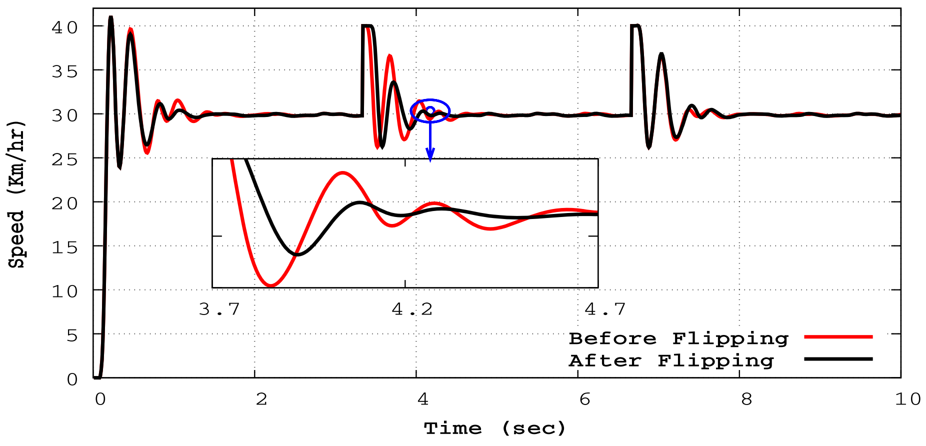
The advantage of the proposed method can be further appreciated by showing the plant's response as given in Figure 11. It shows the output response of CC1 and CC3 with and without slot exchanges between the two loops, as chosen by our methodology corresponding to the initial loop execution patterns given in first entry of Table 2. In this case, CC1 drops an execution slot from its loop execution pattern and CC3 uses the relinquished computation slots to add an execution slot in its loop execution pattern. We used a disturbance signal as the input, consisting of Gaussian state noise with a co-variance $R = 0.005 \times (BB^{\mathsf{T}})$, and a periodic spike arriving every 2 seconds with peak amplitude of 40 km/hr. The response over a time window of 10 seconds is shown in Figure 11. The improvement in the response of CC3 is evident from Figure 11(b), whereas the degradation in the response of CC1 as a consequence of the drop is marginal (see Figure 11(a)).

Table 3 compares the maximum percentage gain from various combinations of features. The number of features and the their combinations are reported in Column 1 and Column 2, respectively. For each such case, the improvement in control cost (i.e., *%Gain*) is computed for a given combination of input loop execution patterns and a static schedule. For each case, the total number of flippable pairs and *%Gain* obtained using our methodology are reported in Column 3 and Column 4, respectively. It may be noted that the efficacy of our approach increases as we pack more features on a shared ECU. For example, the last two rows of Table 3 have five control loops on a shared ECU, and we are able to improve the control cost by more than 30%.

Note that the idea of post-processing of control schedules does not have a direct comparison in existing control design frameworks. However, among existing works, we select the state-of-the-art method proposed in [30] to compare our results. An LMI-based stability result for designing a stable controller for any pattern satisfying a given (m, k)-firm constraint is proposed in [30]. We implement the main stability theorem (i.e., Theorem 4) of [30] and compare the results with our approach for all the nine different initial choices of co-schedulable patterns reported in Table 2.



(a) Output Response of CC1



(b) Output Response of CC3

Fig. 11. Comparison of output responses of CC1 and CC3.

Table 3. *%Gain* versus Choice of Control Features

# Loops	Choice of Features	FP	*%Gain*
3	{CC3, SC1, SC2}	62	23.4
3	{CC1, CC3, SC2}	128	21.35
3	{CC3, SC2, MS2}	227	22.54
4	{CC1, CC2, SC1, MS1}	166	26.74
4	{EB1, EB2, SC1, MS2}	680	28.52
4	{CC1, CC2, SC1, MS2}	789	28.28
4	{CC1, CC2, MS1, MS2}	1443	27.29
5	{CC1, CC2, SC3, MS1, MS2}	1124	33.23
5	{CC1, CC2, SC1, MS1, MS2}	1148	34.78
5	{EB1, CC3, EB2, MS1, MS2}	846	35.07

Table 4. Comparison of Control Cost with [30]

A. For the set {CC1, CC3, SC2}				
ID	Associated (m,k)-firm constraint	**Existing**	**Proposed**	***%Gain***
1.	$\{(14,20),(15,20),(14,20)\}$	60000	16787	72
2.	$\{(12,20),(15,20),(16,20)\}$	103920	50874	51
3.	$\{(14,20),(15,20),(14,20)\}$	103410	79339	23.2
B. For the set {CC1, CC2, SC1, MS1}				
1.	$\{(11,20),(15,20),(12,20),(6,10)\}$	105220	56880	45.97
2.	$\{(11,20),(15,20),(12,20),(5,10)\}$	105420	56885	46
3.	$\{(16,20),(14,20),(15,20),(7,10)\}$	104840	29518	71.8
C. For the set {CC1, CC2, SC3, MS1, MS2}				
1.	$\{(12,20),(12,20),(12,20),(6,10),(6,10)\}$	106610	58364	45.2
2.	$\{(13,20),(12,20),(12,20),(6,10),(6,10)\}$	104032	47206	54.6
3.	$\{(13,20),(12,20),(12,20),(7,10),(6,10)\}$	104310	48450	53.5

The comparative results are presented in Table 4. Column 1 indicates the id for the choices of patterns for a set of control loops. The chosen set of patterns (with id) are the same as reported in Table 2. Column 2 shows the associated (m,k)-firm constraints for these sets of patterns. Note that we implement the stability theorem of [30] for these choices of (m,k)-firm patterns. Column 3 reports the combined control cost obtained by applying the method proposed in [30]. Thereafter, our proposed approach is applied to optimize the control performance for these choices of patterns. Column 4 reports the combined control cost obtained for scheduling patterns synthesized applying our proposed approach. As is evident from Table 4, for each of these choices of patterns, we get much improved combined control performance by applying our method. The percentage improvement in combined control cost is shown in Column 5 of Table 4.

One reason for such inferior control performance by [30] is that the proposed stability theorem of [30] generates one single controller that guarantees closed-loop stability for *any* pattern satisfying the given (m,k)-firm constraint. It is well articulated in the literature [21, 36, 37] that control performance varies with the choice of scheduling patterns based on the distribution of drops over the patterns. The method for stable controller design in [30] considers the worst-case (in terms of control performance) choice of the (m,k)-pattern while solving the LMIs for finding the control gain. The method does not explore multiple control gains and does not target optimization for a given scheduling pattern. In our work, since we start with a given choice of such patterns, one for each control loop and continue optimizing the gains by exchanging scheduling slots, our methodology provides scope for generating more optimized control solutions as is evident from our results.

7 CONCLUSIONS AND FUTURE WORK

The design of embedded controller has traditionally been treated as a two-step process, where the design of the control law for a chosen sampling rate is followed by the scheduling of control tasks on the ECU as per the sampling rate. With the integration of multiple control loops on shared ECUs, loop execution patterns with drops enable the co-scheduling tasks while optimizing the ECU-bandwidth. In this article, we have shown for the first time, that trading the positions of the drops can be an effective mechanism for improving the combined control cost of the set of control loops sharing an ECU. We believe that the demonstrated gains will encourage researchers and practitioners to explore more effective methods for post-processing loop execution schedules. One possible future direction that we plan to pursue is to integrate our method as a back-end optimizer for existing control design and scheduling toolboxes like Truetime [24].

Our post-processing based control scheduling and gain optimization methodology opens up novel opportunities in terms of embedded control design with a shared execution platform. In current practice, designers leave out any scope for future optimizations and conservatively select high sampling periods for schedulability. Since our method incorporates scheduling with dropped patterns, it better utilizes the overall available bandwidth, tradeoff scheduling slots among different controllers, and the like. This encourages a control system designer to choose as low a sample period as possible based on the initial budget of bandwidth allocated for each control loop. The above enables an important design trade-off among whether to choose controller gains with a higher sampling period or go for controller gains with a lower sampling period but injected drop patterns. As is evident, the latter may be useful in many cases as a design solution.

APPENDIX

A A NOTE ON CONTROL SYSTEM MODELING

Though we have used the model as given in Equation (3), our proposed method is independent of the model of the controller. Given any linear discrete time plant and feedback controller models, our post-processing method works perfectly. We could have used any other model, e.g., the standard model of $u[k] = Kx[k]$. For better clarification, the following text discusses the model used in this article and the standard feedback model.

Case I. The model that we have used in this article is also used in many research papers (e.g., [2], [3], and [13]); since many control designers consider the controller as a dynamical system itself with its own state evolving linearly with time. Thus, we have the plant and the controller dynamics as given in Equation (1) and Equation (2), respectively. In Equation (2), the control state, i.e., $x_c[k]$ is nothing but the sampled state of the plant, and hence, the matrix C_c is the control gain (similar to the control gain K in the standard feedback dynamics: $u[k] = Kx[k]$). Combining Equation (1) and Equation (2), we get,

$$\begin{aligned} x_p[k+1] &= A_p x_p[k] + B_p C_c x_c[k] \\ x_c[k+1] &= A_c x_c[k] + B_c C_p x_p[k]. \end{aligned} \tag{11}$$

Hence, defining an augmented state vector $x[k]$, the closed-loop dynamics is

$$x[k+1] = \begin{bmatrix} x_p[k+1] \\ x_c[k+1] \end{bmatrix} = \begin{bmatrix} A_p & B_pC_c \\ B_cC_p & A_c \end{bmatrix} \begin{bmatrix} x_p[k] \\ x_c[k] \end{bmatrix} = \begin{bmatrix} A_p & B_pC_c \\ B_cC_p & A_c \end{bmatrix} x[k].$$

Now, in drop scenario at $(k+1)$-th sample, following the hold strategy the controller dynamics becomes

$$x_c[k+1] = x_c[k] = Ix_c[k] + Ox_p[k].$$

Thus, the closed-loop model becomes

$$x[k+1] = \begin{bmatrix} x_p[k+1] \\ x_c[k+1] \end{bmatrix} = \begin{bmatrix} A_p & B_pC_c \\ O & I \end{bmatrix} \begin{bmatrix} x_p[k] \\ x_c[k] \end{bmatrix} = \begin{bmatrix} A_p & B_pC_c \\ O & I \end{bmatrix} x[k]. \tag{12}$$

Case II. If we use the control dynamics: $u[k] = Kx[k]$, the closed-loop model becomes

$$x[k+1] = A_p x[k] + B_p K x[k] = (A_p + B_p K) x[k].$$

Using this model, in the drop scenario following the hold strategy, the controller dynamics is

$$u[k] = u[k-1] = Kx[k-1].$$

Hence, the closed-loop dynamics for the drop scenario at $(k+1)$-th sample is as follows:

$$x[k+1] = A_p x[k] + B_p K x[k-1].$$

Note that the state value at $(k+1)$-th sample depends on the state valuation at both the kth and $(k-1)$-th samples. Hence, in order to model the closed-loop system, we need to define an augmented state vector as:

$$X[k] = \begin{bmatrix} x[k] \\ x[k-1] \end{bmatrix}.$$

Therefore, the closed-loop dynamics is

$$\begin{aligned} x[k+1] &= A_p x[k] + B_p K x[k-1] \\ x[k] &= I x[k] + O x[k-1]. \end{aligned}$$

Or,

$$X[k+1] = \begin{bmatrix} x[k+1] \\ \\ x[k] \end{bmatrix} = \begin{bmatrix} A_p & B_p K \\ \\ I & O \end{bmatrix} \begin{bmatrix} x[k] \\ \\ x[k-1] \end{bmatrix} = \begin{bmatrix} A_p & B_p K \\ \\ I & O \end{bmatrix} x[k]. \tag{13}$$

Note that both Equation (12) and Equation (13) represent dynamical systems with the same set of closed-loop poles. As is evident, our post-processing framework can be useful for both these modeling paradigms of controller dynamics.

ACKNOWLEDGMENTS

The first author gratefully acknowledge Dr. Souradeep Dutta and Dr. Sukanta Bhattacharjee for their help at the early stage of this work.

REFERENCES

[1] [n.d.]. http://www.ibm.com/software/integration/optimization/cplex/.
[2] R. Alur, A. D'Innocenzo, K. H. Johansson, G. J. Pappas, and G. Weiss. 2011. Compositional modeling and analysis of multi-hop control networks. *IEEE Trans. Automat. Control* 56, 10 (Oct 2011), 2345–2357.
[3] Rajeev Alur and Gera Weiss. 2008. Regular specifications of resource requirements for embedded control software. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*. 159–168.
[4] Karl J. Åström and Björn Wittenmark. 1997. *Computer-Controlled Systems*. Prentice-Hall, Inc.
[5] G. Bernat, A. Burns, and A. Liamosi. 2001. Weakly hard real-time systems. *IEEE Trans. Comput.* 50, 4 (Apr. 2001), 308–321.
[6] R. Blind and F. Allgöwer. 2015. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *Proceedings of the Conference on Decision and Control*. 7510–7515.
[7] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. 1994. *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics.
[8] Rosa Castané et al. 2006. Resource management for control tasks based on the transient dynamics of closed-loop systems. In *Proceedings of the Euromicro Conference on Real-Time Systems*. 10–pp.
[9] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. 2002. Feedback feedforward scheduling of control tasks. *Real-Time Systems* 23, 1–2 (2002), 25–53.
[10] Anton Cervin, Manel Velasco, Pau Marté, and Antonio Camacho. 2009. *Optimal on-line sampling period assignment*. Department of Automatic Control, Technical University of Catalonia, Barcelona, Spain, Technical Report ESAII-RR-09-04 (2009).
[11] Anton Cervin, Manel Velasco, Pau Marté, and Antonio Camacho. 2011. Optimal online sampling period assignment: Theory and experiments. *IEEE Transactions on Control Systems Technology* 19, 4 (2011), 902–910.
[12] Wanli Chang, Dip Goswami, Samarjit Chakraborty, and Arne Hamann. 2018. OS-aware automotive controller design using non-uniform sampling. *ACM Transactions on Cyberphysical Systems* 2, 4 (July 2018), 26:1–26:22.
[13] A. D'Innocenzo, M. D. Di Benedetto, and E. Serra. 2011. Link failure detection in multi-hop control networks. In *Proceedings of the Conference on Decision and Control and European Control Conference*. 5248–5253.
[14] Johan Eker, Per Hagander, and Karl-Erik Årzén. 2000. A feedback scheduler for real-time controller tasks. *Control Engineering Practice* 8, 12 (2000), 1369–1378.

[15] Felicioni Flavia et al. 2008. Optimal on-line (*m, k*)-firm constraint assignment for real-time control tasks based on plant state information. In *Proceedings of the Conference on Emerging Technologies and Factory Automation.* 908–915.

[16] Simon Fürst. 2015. Autosar the next generation–the adaptive platform. *CARS@ EDCC2015* (2015).

[17] Michael Garey and David S. Johnson. 1979. *Computers and Intractability.* W. H. Freeman and Company, New York.

[18] Maximilian Gaukler, Tim Rheinfels, Peter Ulbrich, and Günter Roppenecker. 2019. Convergence Rate Abstractions for Weakly-Hard Real-Time Control. arxiv:1912.09871

[19] S. Ghosh, S. Dey, and P. Dasgupta. 2018. Co-synthesis of loop execution patterns for multihop control networks. *IEEE Embedded System Letters* 10, 4 (Dec. 2018), 111–114.

[20] Sumana Ghosh, Soumyajit Dey, and Pallab Dasgupta. 2019. Performance and energy aware robust specification of control execution patterns under dropped samples. *IET Computers & Digital Techniques* (June 2019).

[21] Sumana Ghosh, Souradeep Dutta, Soumyajit Dey, and Pallab Dasgupta. 2017. A structured methodology for pattern based adaptive scheduling in embedded control. *ACM Transactions on Embedded Computing Systems* 16, 5s (Sept. 2017), 189:1–189:22.

[22] Dip Goswami, Alejandro Masrur, Reinhard Schneider, Chun J. Xue, and Samarjit Chakraborty. 2013. Multirate controller design for resource-and schedule-constrained automotive ECUs. In *Proceedings of the Design, Automation and Test in Europe Conference.* 1123–1126.

[23] Dan Henriksson and Anton Cervin. 2005. Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In *Proceedings of the Conference on Decision and Control.* 4469–4474.

[24] Dan Henriksson, Anton Cervin, and Karl-Erik Årzén. 2002. TRUETIME: Simulation of control loops under shared computer resources. *IFAC Proceedings, Volume* 35, 1 (2002), 417–422.

[25] Chao Huang, Wenchao Li, and Qi Zhu. 2019. Formal verification of weakly-hard systems. In *Proceedings of the Hybrid Systems: Computation and Control.* 197–207.

[26] Ning Jia, Ye-Qiong Song, and Rui-Zhong Lin. 2006. Analysis of networked control system with packet drops governed by (m,k)-firm constraint*. In *Fieldbus Systems and Their Applications 2005.* 63–70.

[27] Florian Kluge, Markus Neuerburg, and Theo Ungerer. 2015. Utility-based scheduling of (*m*, *k*)-firm real-time task sets. In *Architecture of Computing Systems (ARCS'15).* 201–211.

[28] Jin Ho Kwak and Sungpyo Hong. 2004. *Linear Algebra.* Springer.

[29] Daniel Liberzon. 2003. *Switching in Systems and Control.* Springer.

[30] S. Linsenmayer and F. Allgower. 2017. Stabilization of networked control systems with weakly hard real-time dropout description. In *Proceedings of the Conference on Decision and Control.* 4765–4770.

[31] Chung Laung Liu and James W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* 20, 1 (1973), 46–61.

[32] Rupak Majumdar, Indranil Saha, and Majid Zamani. 2011. Performance-aware scheduler synthesis for control systems. In *Proceedings of the Conference on Embedded Software.* 299–308.

[33] Patrizia Marti et al. 2009. Draco: Efficient resource management for resource-constrained control tasks. *IEEE Trans. Comput.* 58, 1 (2009), 90–105.

[34] M. Mizuochi, T. Tsuji, and K. Ohnishi. 2007. Multirate sampling method for acceleration control system. *IEEE Transactions on Industrial Electronics* 54, 3 (June 2007), 1462–1471.

[35] Miad Moarref and Luis Rodrigues. 2013. Exponential stability and stabilization of linear multi-rate sampled-data systems. In *Proceedings of the American Control Conference*, 158–163.

[36] Jia Ning, Song YeQiong, and Simonot-Lion Francoise. 2007. Graceful degradation of the quality of control through data drop policy. In *Proceedings of the European Control Conference.* 4324–4331.

[37] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. 2018. Beyond the weakly hard model: Measuring the performance cost of deadline misses. In *Proceedings of the Euromicro Conference on Real-Time Systems*, Vol. 106. 10:1–10:22.

[38] Parameswaran Ramanathan. 1999. Overload management in real-time control applications using (*m,k*)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems* 10, 6 (1999), 549–559.

[39] Debayan Roy, Licong Zhang, Wanli Chang, Dip Goswami, and Samarjit Chakraborty. 2016. Multi-objective co-optimization of FlexRay-based distributed control systems. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium.* 1–12.

[40] Indranil Saha, Sanjoy Baruah, and Rupak Majumdar. 2015. Dynamic scheduling for networked control systems. In *Hybrid Systems: Computation and Control.* ACM, 98–107.

[41] Shuichi Sakai, Mitsunori Togasaki, and Koichi Yamazaki. 2003. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics* 126, 2 (2003), 313–322.

[42] Michael Schinkel, Wen-Hua Chen, and Anders Rantzer. 2002. Optimal control for systems with varying sampling rate. In *Proceedings of the American Control Conference*, Vol. 4. 2979–2984.

[43] Danbing Seto, John P. Lehoczky, Lui Sha, and Kang G. Shin. 1996. On task schedulability in real-time control systems. In *Proceedings of the Real-Time Systems Symposium*. 13–21.

[44] Damoon Soudbakhsh et al. 2013. Co-design of control and platform with dropped signals. In *Proceedings of the International Conference on Cyber-Physical Systems*. 129–140.

[45] Gera Weiss and Rajeev Alur. 2007. Automata based interfaces for control and scheduling. In *Proceedings of the Conference on Hybrid Systems: Computation and Control*. 601–613.

[46] Majid Zamani, Soumyajit Dey, Sajid Mohamed, Pallab Dasgupta, and Manuel Mazo. 2016. Scheduling of controllers update-rates for residual bandwidth utilization. In *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems*. 85–101.

Received March 2020; revised July 2020; accepted August 2020