OS-Aware Automotive Controller Design Using Non-Uniform Sampling

WANLI CHANG, Singapore Institute of Technology
DIP GOSWAMI, Eindhoven University of Technology
SAMARJIT CHAKRABORTY, Technical University of Munich
ARNE HAMANN, Robert Bosch GmbH

Automotive functionalities typically consist of a large set of periodic/cyclic tasks scheduled under a real-time operating system (OS). Many of the tasks are feedback control applications with stringent performance requirements. OSEK/VDX is a common class of automotive OS that offers preemptive periodic schedules supporting a pre-configured set of periods. The feedback controllers implemented onto such OSEK/VDX-compliant systems need to use one of the pre-configured (sampling) periods. A shorter period is often desired for a higher control performance, and this implies a higher processor load. For a given performance requirement, the longest sampling period that meets this requirement is the optimal one. Given a limited set of pre-configured periods, such optimal sampling periods are often not available, and the practice is to choose a shorter available period—leading to a higher processor load. To address this, we propose a controller that cyclically switches among the available periods, thereby leading to an average sampling period closer to the optimal one. This way, we reduce the processor load and are able to pack more control applications on the same processor. The main challenge in this article is the design of such controllers that takes into account such cyclic switching of sampling periods (i.e., use non-uniform sampling). The controller needs to meet specified performance requirements (settling time) and system constraints (e.g., input saturation). Such a non-convex constrained controller optimization problem as raised in the OS-aware automotive systems design has not been addressed in the traditional optimal control literature. A novel approach based on adaptively parameterized particle swarm optimization (PSO) is proposed to solve it. Using the OS-aware controller design with non-uniform sampling, we show that a higher number of applications can be packed on a processor, which is of particular interest in the cost-sensitive automotive industry.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; **Real-time operating systems**;

Additional Key Words and Phrases: OSEK/VDX, non-uniform sampling, computation resources, optimal control

ACM Reference format:
Wanli Chang, Dip Goswami, Samarjit Chakraborty, and Arne Hamann. 2018. OS-Aware Automotive Controller Design Using Non-Uniform Sampling. *ACM Trans. Cyber-Phys. Syst.* 2, 4, Article 26 (July 2018), 22 pages.
https://doi.org/10.1145/3121427

This work is partially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) program.
Authors' addresses: W. Chang, Singapore Institute of Technology, 10 Dover Drive, Singapore 138683; email: wanli.chang@singaporetech.edu.sg; D. Goswami, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands; email: d.goswami@tue.nl; S. Chakraborty, Technical University of Munich, Arcisstrasse 21, D-80290, Munich, Germany; email: samarjit@tum.de; A. Hamann, Robert Bosch GmbH, Renninger, 70465 Stuttgart, Germany; email: arne.hamann@de.bosch.com.

https://doi.org/10.1145/3121427

1 INTRODUCTION

In the past decade, the complexity of automotive software and the number of applications and software tasks have considerably increased. For instance, in engine management, the main drivers are complex exhaust gas treatment systems like NOx storage catalyst converter (NSC) and selective catalytic reduction (SCR) and fuel efficiency measures like staged injection and variable cam timing (Jeong et al. 2011; Popovic et al. 2003). As a consequence, standard engine management software nowadays contains dozens of tasks with around 1,500 runnables (Kramer et al. 2015). At the same time, automotive systems are highly cost sensitive, and there is an increasing effort to integrate multiple tasks onto a single electronic control unit (ECU). In line with such developments, in this article we study a commonly occurring setup—where multiple feedback control applications are to be implemented on a single ECU. The goal is to pack as many applications as possible in an effort to reduce costs.

Tasks in automotive software systems are typically scheduled with preemptive policies and cyclically repeated with a fixed period on OSEK/VDX-compliant operating systems (OS)[1] (Feiler 2003; Consortium 2005; Apuzzo et al. 2016). For control applications, runnables containing the functional code are assigned to tasks according to the continuous dynamics of the physical process being controlled. For example, injection control in engine management has faster dynamics than exhaust gas control and thus requires a shorter period.

A feedback control loop consists of three operations:

- **Measure:** Sensors measure the states of the physical plants. This is also called sampling.
- **Compute:** Taking the data from sensors, control programs are executed and compute the control input.
- **Actuate:** The control input is sent to actuators, aiming to achieve certain desired behavior of the plants.

In this work, we assume that the measure and the actuate operations take negligible time compared to the compute operation, and they are performed in a separate sensing/actuating unit under a strict time-triggered policy. As shown in Figure 1, the time duration between two consecutive measurements (or samplings) of the plant states is defined as the sampling period h. The time duration between the measurement and the actuation of one feedback control loop is defined as the sensor-to-actuator delay τ^{sa}. The actual execution time of the control program is denoted as E and the worst-case execution time (WCET) is E^{wc}. The actuate operation is performed exactly after E^{wc} time from the measure operation while the compute operation is performed in between. This setting leads to a constant sensor-to-actuator delay, i.e., $\tau^{sa} = E^{wc}$.

Generally, a shorter sampling period allows the controller to respond to its plant more frequently and is thus potentially able to achieve a better control performance with an appropriately designed controller. The obvious downside is a higher processor load, since the control program is executed more frequently. This prevents more functions and applications from being integrated onto the ECU. Therefore, the controller should be designed to use the largest possible sampling period (to reduce ECU load) that is able to fulfill the control performance requirement and satisfy the system constraints. This is the *optimal* sampling period that should be ideally used.

[1]Open Systems and Their Corresponding Interfaces for Automotive Electronics (OSEK) is a joint project in the German automotive industry founded in 1993 with initial partners of BMW, Bosch, DaimlerChrysler, Opel, Siemens, and Volkswagen. It was later joined by the French car manufacturers PSA and Renault introducing their Vehicle Distributed Executive (VDX) approach. The goal is to define an industry standard for an open-ended architecture for distributed control units in vehicles.

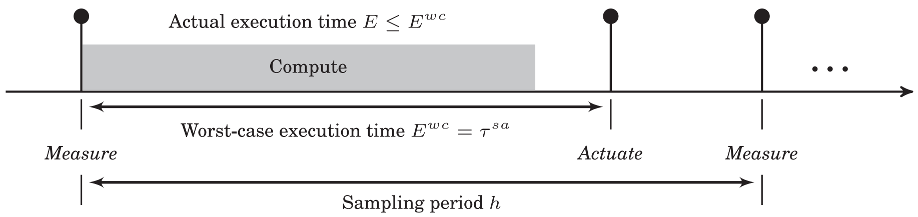
Actual execution time $E \leq E^{wc}$
Compute
Worst-case execution time $E^{wc} = \tau^{sa}$
Measure
Actuate
Measure
Sampling period h

Fig. 1. The general timing model of a control loop.

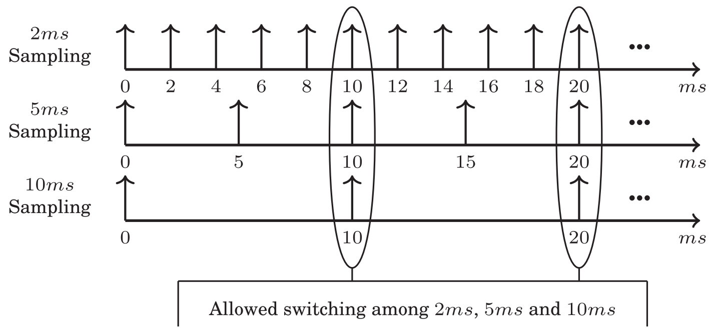

Fig. 2. Allowed switching instants among multiple sampling periods.

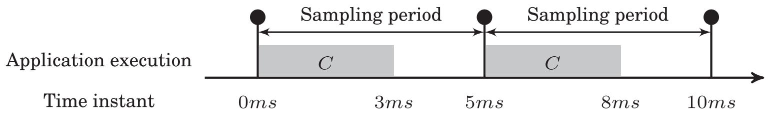

Fig. 3. Packing of control applications onto the ECU.

However, an OSEK/VDX OS is usually pre-configured to support a small set of predefined sampling periods.[2] Hence, often the optimal sampling period is not directly realizable on the ECU. The conventional way to handle it is to use the largest sampling period from the pre-configured set of sampling periods available in the OSEK/VDX OS that is shorter than the optimal one. It is clearly a waste of scarce computation resources on board.

Main idea: In this work, we design controllers that *switch* between the available pre-configured sampling periods offered by the OSEK/VDX OS, following a predefined static schedule. A typical example with sampling periods of 2, 5, and 10ms on OSEK/VDX OS is illustrated in Figure 2. For one application, switching between two sampling periods can only occur at the common multiplier of them. For instance, switching between 2 and 5ms is possible at the time instant of 10ms, 20ms, and so on. Therefore, possible sequences of sampling periods are {2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms, repeat}, {5ms, 5ms, 10ms, repeat}, and so on.

Illustrative example: We now explain a simple case that multiple identical control applications C need to be implemented on ECUs. Assuming that the control performance requirement of C can be satisfied with a sampling period of 5ms, yet not with 10ms. If the WCET of C is 3ms, then only one application can be implemented on the ECU as shown in Figure 3, since the sampling

[2]Theoretically, more periods can be created. However, due to the large number of software runnables from many different suppliers, this will create much overhead and is thus practically infeasible.

period 5ms is not long enough to execute two applications, which require 6ms. If 10ms is used as the sampling period, then three applications can be integrated into the ECU. However, this is not feasible due to the violation of the control performance requirement. A non-uniform sampling schedule {5ms, 5ms, 10ms, repeat} achieves an average sampling period of 6.67ms, which reduces the processor load compared to the schedule with a constant sampling period of 5ms and enables two applications to be packed onto the ECU. Detailed scheduling will be explained later in this article. The question is how to design the controller for such a non-uniform sampling schedule to satisfy the control performance requirement.

Contributions: The main technical challenge we address is designing a controller that uses a non-uniform scheme, striving for optimizing control performance and respecting system constraints simultaneously to pack more control applications on one processor. Given a non-uniform sampling schedule, which could come up by checking the performance of uniform sampling schedules, our proposed controller design approach optimizes the settling time—a common control performance metric that is especially important for real-time applications and harder to analyze than quadratic cost and explicitly respects the hard physical constraint on the input saturation. Such a constrained non-convex optimization problem with significant non-linearity has not been addressed in the control theory literature and does not lend itself to an analytical closed-form solution. Therefore, one has to resort to heuristic optimization techniques. In this article, we address this problem using an approach based on particle swarm optimization (PSO) with adaptive parameterization for controller pole-placement. The proposed idea is evaluated on a real-life electro-mechanical braking (EMB) system. The number of applications implemented on an ECU can be higher, which makes the presented approach attractive for the cost-sensitive automotive domain.

Although the OS-related constraints are a major problem faced in the industry when designing embedded control systems, currently there are no systematic solutions. This is the first article that provides a solution to handle OS-related characteristics directly in the control strategy. While there have been works taking network characteristics or communication resources into account when designing controllers (Hong et al. 2010, 2015; Chen et al. 2014), in many embedded systems, computation resources are often also scarce (due to the use of simple microcontrollers and cost pressures). Our work goes in this direction and the techniques we provide result in computation-resource-efficient controllers.

Organization: The rest of the article is organized as follows. Section 2 discusses the related work on resource-aware embedded control systems design, optimal control and non-uniform sampling. Section 3 describes the automotive system architecture under consideration, including feedback control applications and the OSEK/VDX OS. The OS-aware controller design is presented in Section 4. A novel PSO technique with adaptive parameterization is proposed in Section 5 to solve the pole-placement problem. Section 6 introduces an alternative controller design with better scalability. Experimental results on the EMB system are reported in Section 7, and Section 8 makes the concluding remarks.

2 RELATED WORK

There have been a number of works on resource-aware embedded control systems design, most of which consider the communication among networked systems (Anta and Tabuada 2009; Yue et al. 2013; Roy et al. 2016). The conventional paradigm in networked embedded control systems regards the messages as periodic, which facilitates the analysis and implementation yet leads to conservative usage of the communication bandwidth. An aperiodic strategy for dynamic allocation of bandwidth according to the current state of the plants and available resources is proposed in Anta and Tabuada (2009). Control loops closed over Controller Area Network (CAN) are discussed and illustrated on a train car. In Yue et al. (2013), a delay system model is constructed by investigating the effect of the network transmission delay. A novel event-trigger controller that co-designs the feedback gain and the trigger parameters is proposed and shown to have superior performance with simulation results. A co-optimization approach, which synthesizes both the controllers and communication schedules for FlexRay-based distributed control systems, is reported in Roy et al. (2016). The scalability issue is addressed and the tradeoff between control performance and bus utilization offers flexibility to designers.

Computation resources have been taken into account in only a few works (Castane et al. 2006; Greco et al. 2011; Cervin et al. 2002). A resource management strategy adjusting the task periods at runtime considering the response over a finite time horizon of the plants is proposed in Castane et al. (2006) to maximize the control performance. In Greco et al. (2011), a novel controller design technique based on a hierarchy of controllers is proposed, so that when the allocated execution time is short, a low-level computationally light controller is activated to achieve basic control performance and when the execution time is long, a high-level computationally intensive controller is used aiming for better control performance. In Cervin et al. (2002), a scheduling architecture for real-time control tasks is proposed. The scheduler uses feedback from execution time measurements and feedforward from workload changes to adjust the sampling periods of the control tasks so that the combined performance (a linear or quadratic cost function) of the controllers is optimized. None of the efforts above address the restriction from the OS.

Works in control theory literature with non-uniform sampling and switched systems focus on guaranteeing stability (Lin and Antsaklis 2009; Lemmon and Hu 2011). Generally, theoretical tools such as common quadratic Lyapunov functions (CQLF) and switched Lyapunov functions (SLF) tackle arbitrary switching between sampling periods to assure stability of the overall closed-loop system. In our work, as opposed to arbitrary switching, the sequence of sampling periods is cyclic and decided in the design phase. We aim for further performance optimality by exploiting this additional knowledge about the switching behavior. In the field of optimal control, techniques such as linear quadratic regulator (LQR) and its variants (e.g., periodic LQR) (Lavretsky and Wise 2013) are well developed. However, they do not explicitly consider the hard constraint on the control input, which exists in most real-life control applications such as automobiles.

The combination of performance optimization and input constraint satisfaction is addressed by model predictive control (MPC) techniques (Rawlings and Mayne 2009)—another well-developed area. MPC performs online optimization in every sampling period, making it computationally heavy and unsuitable for being implemented on the resource-constrained embedded platform, which is what we are studying in this work. Explicit MPC pre-optimizes the controller for all possible system states and searches for the optimal control input from a look-up table online. These existing optimal control methods cannot be directly applied in our work, since their optimization objective is quadratic cost. Considering both settling time and input saturation simultaneously, which are non-convex on the controller poles is particularly challenging and will be addressed in this work.

Some recent and notable efforts in the research of optimal control and non-uniform sampling include (Bini and Buttazzo 2014) and Cervin et al. (2011). The optimal sampling problem is tackled in Bini and Buttazzo (2014), where the sampling instants and control inputs are selected to minimize a given function of the system state and control input. In particular, a necessary condition for the optimality of a set of sampling instants is derived and a quantization-based sampling strategy is proposed to be computationally tractable. However, the sampling periods in Bini and Buttazzo (2014) can be arbitrarily chosen and no constraints (e.g., from the OS as we are studying) are taken into account. In addition, the proposed method is only applied in the LQR problem, which is relatively simple to analyze due to its quadratic cost function.

Online optimal sampling period assignment is investigated in Cervin et al. (2011) to maximize the control performance. A feedback scheduler is developed to periodically assign new sampling periods based on the current plant states. It is shown that most computation can be done offline and stored in a look-up table. Again, only the quadratic cost function is considered and the selection of sampling periods is not restricted. Besides, since the switching is not fixed, yet occurs depending on the plant states in real time, stability cannot be guaranteed. Building on these previous works, our method formulates an optimal pole-placement problem for a non-uniform schedule known in the design phase, where the input saturation is explicitly respected, the settling time is minimized, and the stability is ensured.

3 SYSTEM ARCHITECTURE

We consider a typical automotive setup that multiple feedback control applications share an ECU with other applications. The ECU runs OSEK/VDX OS. The two main elements of such a system architecture—feedback control applications and the OSEK/VDX OS—are described in this section.

3.1 Feedback Control Applications

Plant dynamics: A control application is responsible for controlling a plant or a dynamic system. In particular, we consider linear single-input-single-output (SISO) control applications where the dynamic behavior is modeled by a set of differential equations,

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t), \\ y(t) &= \mathbf{C}\mathbf{x}(t), \end{aligned} \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ is the *system state*, $\dot{\mathbf{x}}(t)$ is the derivative of $\mathbf{x}(t)$ with respect to time, $y(t)$ is the *system output*, and $u(t)$ is the *control input* applied to the system. The number of system states is n. The *system matrix* (or *state matrix*) is $\mathbf{A}$, the *input matrix* is $\mathbf{B}$, and the *output matrix* is $\mathbf{C}$. These matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ capture physical properties of the plant. System poles are eigenvalues of $\mathbf{A}$.

Discretized dynamics: In most applications, the controller is implemented in a digital fashion on a computer. This implies that the system states must be sampled when measured by the sensors, as has been shown in Figure 1. Assuming the sampling period to be h, the sampled system state is denoted as

$$\mathbf{x}[k] = \mathbf{x}(t_k), \quad t_k = kh, \; k = 0, 1, 2, 3, \ldots. \tag{2}$$

Similarly, the sampled system output is

$$y[k] = y(t_k). \tag{3}$$

The control input taking discrete values is denoted as $u[k]$, which is passed through a zero-order hold (ZOH)[3] and applied to the plant. The output of the ZOH is given by

$$u(t) = u[k], \quad t_k \leq t < t_{k+1}. \tag{4}$$

The discretized dynamics can then be derived by solving Equation (1) (Åström and Murray 2009),

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d u[k], \\ y[k] &= \mathbf{C}x[k], \end{aligned} \tag{5}$$

where

$$\mathbf{A}_d = e^{\mathbf{A}h}, \; \mathbf{B}_d = \int_0^h (e^{\mathbf{A}\tau'} d\tau')\mathbf{B}. \tag{6}$$

Clearly, $\mathbf{A}_d$ and $\mathbf{B}_d$ depend on the sampling period h.

[3]ZOH converts a discrete-time signal to a continuous signal by holding each sample value for the entire sampling period.

System controllability: Controllability of a discrete system is defined as the ability to transfer the system from any initial state $\mathbf{x}[0] = \mathbf{x}_0$ to any desired final state $\mathbf{x}[k_f] = \mathbf{x}_f$ in a finite time. The square controllability matrix is

$$\mathcal{CO} = \begin{bmatrix} \mathbf{B}_d & \mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{A}_d^{n-1}\mathbf{B}_d \end{bmatrix}. \tag{7}$$

If $\mathcal{CO}$ is non-singular, then there is a unique sequence of control input that transfers the initial system state $\mathbf{x}_0$ to the desired system state $\mathbf{x}_f$ in k_f steps. In this work, we require the system to be controllable.

Control performance: Settling time is a widely used metric to quantify the control performance, especially for real-time control applications. In general, settling time in control systems is the time required for a response to become steady. In this work, it is defined as the time for the system output $y[k]$ to reach and stay in a closed region around the reference value r (e.g., $0.98r$ to $1.02r$) and denoted as t_s. Shorter settling time implies better control performance. For safety-critical applications, there is often a requirement that the settling time must be within a certain period t_s^0.

Input saturation: In almost every real-world system, due to the physical constraint of the actuator, there is a maximum available control input (e.g., the maximum voltage or current that can be supplied by a battery) and the controller needs to be designed such that the maximum value of $|u[k]|$ does not exceed this limit U_{max}, i.e., $|u[k]| \leq U_{max}$. This is the constraint of the input saturation.

State-feedback control: The general structure of a linear state-feedback controller is as follows:

$$u[k] = \mathbf{K}\mathbf{x}[k] + Fr, \tag{8}$$

where $\mathbf{K}$ is the feedback gain and F is the feedforward gain. Then, the closed-loop system dynamics becomes

$$\mathbf{x}[k+1] = (\mathbf{A}_d + \mathbf{B}_d\mathbf{K})\mathbf{x}[k] + \mathbf{B}_d Fr, \tag{9}$$

Pole-placement: Different locations of closed-loop system poles, i.e., eigenvalues of $(\mathbf{A}_d + \mathbf{B}_d\mathbf{K})$, result in different system behaviors. In pole-placement, poles are placed in desired locations (eigenvalues are set) often to fulfill various high-level goals, such as control performance maximization and system constraints satisfaction. The desired poles p can be decided with empirical or optimization techniques. This method is feasible, since there is the freedom to choose the feedback gain $\mathbf{K}$. All eigenvalues of $(\mathbf{A}_d + \mathbf{B}_d\mathbf{K})$ must have absolute values of less than unity to ensure system stability (Åström and Murray 2009). In this work, we formulate the pole-placement as a constrained optimization problem, with the poles as decision variables, the control performance as the optimization objective, and system requirements (input saturation and system stability) as constraints. The maximum control performance (i.e., the minimum settling time) is then checked against its requirement. The number of poles (i.e., the number of decision variables) is equal to the number of system states n. A novel PSO-based technique is proposed to solve this challenging non-convex optimization problem.

Feedback and feedforward gain: Once the pole locations are decided, the following characteristics equation of z can be constructed with these poles as roots:

$$z^n + \gamma_1 z^{n-1} + \gamma_2 z^{n-2} + \cdots + \gamma_n = 0. \tag{10}$$

Substituting the n roots into (10) results in n simultaneous equations that can be solved to obtain $\gamma_1, \gamma_2, \ldots, \gamma_n$. Then we define

$$\gamma_c(\mathbf{A}_d) = \mathbf{A}_d^n + \gamma_1\mathbf{A}_d^{n-1} + \gamma_2\mathbf{A}_d^{n-2} + \cdots + \gamma_n\mathbf{I}, \tag{11}$$

Table 1. An Example OSEK/VDX OS Time Table of Applications Release

Time instant	Release
0ms	Applications with periods of 2ms, 5ms, and 10ms
2ms	Applications with the period of 2ms
4ms	Applications with the period of 2ms
5ms	Applications with the period of 5ms
6ms	Applications with the period of 2ms
8ms	Applications with the period of 2ms
10ms	**Repeat actions at 0ms**

where $\mathbf{I}$ is an identity matrix. According to Ackermann's formula (Ackermann and Utkin 1998), the feedback gain used to stabilize the closed-loop system (i.e., to continuously make $\mathbf{x}[k+1]$ approach $\mathbf{x}[k]$) is calculated as

$$\mathbf{K} = [0 \quad \cdots \quad 0 \quad 1]\, CO^{-1}\, \gamma_c(\mathbf{A}_d). \tag{12}$$

Note that a stable system will have $x[k+1] = x[k]$ and $y[k] = Cx[k]$ in the steady-state. The static feedforward gain F is used to make the system output $y[k]$ track the reference r. We let $\mathbf{x}[k+1] = \mathbf{x}[k]$ and $y[k] = \mathbf{C}\mathbf{x}[k] = r$. Rearranging Equation (9), leads to

$$F = \frac{1}{\mathbf{C}_d(\mathbf{I} - \mathbf{A}_d - \mathbf{B}_d\mathbf{K})^{-1}\mathbf{B}_d}. \tag{13}$$

3.2 Operating System

As a class of real-time OS widely used in the automotive industry, OSEK/VDX OS supports preemptive fixed-priority scheduling. That is, priorities are assigned to applications and at any point in time, the task with the highest priority among all active ones is executed.

On OSEK/VDX OS, tasks can be triggered by events (e.g., interrupts) or by time (alarm periods for task activation). The latter scheme is considered in this work, where each application gets released and is allowed to access the processor periodically. There are various periods of release times and each application is assigned one. Different applications may have different periods. Every time an application is released, its program gets the chance to be executed, depending on its priority.

A time table containing all the periodic release times within the alleged *hyperperiod* (i.e., the minimum common multiple of all periods) needs to be configured. An example with a set of three periods 2ms, 5ms, and 10ms is illustrated in Table 1. The hyperperiod is equal to 10ms and the time table repeats itself every 10ms by reseting the timer. The assigned priority will determine the execution order of applications. A higher priority is typically assigned to the application released with a shorter period (i.e., the rate monotonic scheduling), since this generally results in a more efficient use of the processor. To be computationally efficient in practice, the priority is often not fixed and given to the task with the earliest deadline, which is the dynamic earliest deadline first (EDF) scheduling (Buttazzo and Gai 2006). It is to be noted that the approach proposed in this work is orthogonal to the scheduling policy.

An example with two applications C_1 and C_2 sharing one ECU is illustrated in Figure 4. C_1 has a period of 2ms and C_2 has a period of 5ms. The execution time of C_1 is assumed to be 0.7ms and the execution time of C_2 is assumed to be 2ms. C_1 has a higher priority than C_2. Within a hyperperiod of 10ms, C_1 is released at 0ms, 2ms, 4ms, 6ms, 8ms, and 10ms. C_2 is released at 0ms, 5ms, and 10ms. It can be seen that C_2 is executed only when C_1 does not require to access the ECU. For instance, at 0ms, both C_1 and C_2 are released and require access to the ECU. C_1 is permitted to be executed

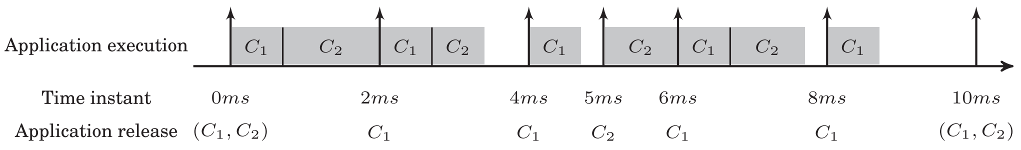

Fig. 4. Release and execution time of two applications sharing one ECU. C_1 with a sampling period of 2ms has a higher priority than C_2 with a sampling period of 5ms. Execution times of C_1 and C_2 are 0.7ms and 2ms, respectively.

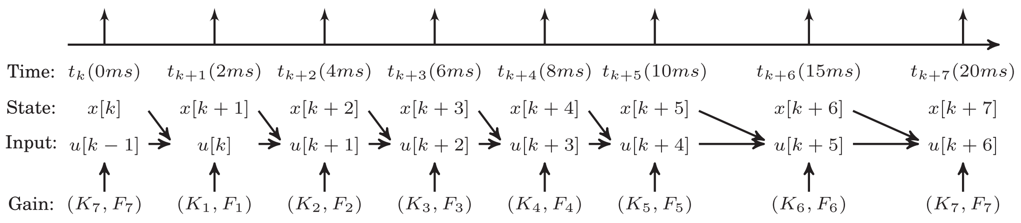
Time: $t_k(0ms)$ $t_{k+1}(2ms)$ $t_{k+2}(4ms)$ $t_{k+3}(6ms)$ $t_{k+4}(8ms)$ $t_{k+5}(10ms)$ $t_{k+6}(15ms)$ $t_{k+7}(20ms)$
State: $x[k]$ $x[k+1]$ $x[k+2]$ $x[k+3]$ $x[k+4]$ $x[k+5]$ $x[k+6]$ $x[k+7]$
Input: $u[k-1]$ $u[k]$ $u[k+1]$ $u[k+2]$ $u[k+3]$ $u[k+4]$ $u[k+5]$ $u[k+6]$
Gain: (K_7, F_7) (K_1, F_1) (K_2, F_2) (K_3, F_3) (K_4, F_4) (K_5, F_5) (K_6, F_6) (K_7, F_7)

Fig. 5. Illustration of the controller design with an example schedule $S^0 = \{2\text{ms}, 2\text{ms}, 2\text{ms}, 2\text{ms}, 2\text{ms}, 5\text{ms}, 5\text{ms}\}$. The actuation occurs at the end of a sampling period. The figure is not drawn to scale.

while C_2 has to wait. At 0.7ms, C_1 completes its execution and C_2 gets the access to the ECU. At 2ms, C_1 starts its execution. Although C_2 has not completed its execution, it is preempted and suspended.

Feedback control under OSEK/VDX OS: As can be seen from the above example, in the preemptive scheduling of OSEK/VDX OS, the execution completion time of the control program varies in different sampling periods. Therefore, we postpone the actuation to the end of a sampling period, i.e., the sensor-to-actuator delay is equal to one sampling period, to avoid varying sensor-to-actuator delays. Referring to Equation (8) and as illustrated in Figure 5, the control input $u[k]$ computed based on the system state $\mathbf{x}[k]$ sampled at the time instant t_k, is applied to the plant at the time instant t_{k+1}. $\mathbf{x}[k+1]$ is then dependent on its previous state $\mathbf{x}[k]$ and the control input $u[k-1]$, which is computed based on $\mathbf{x}[k-1]$ and applied at t_k. It is noted that in this work, the control input $u[k]$ is also dependent on the previous control input $u[k-1]$, which will be explained later in Section 4. The system dynamics in Equation (5) becomes

$$\mathbf{x}[k+1] = \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_d u[k-1]. \tag{14}$$

Processor Load: We assume that the set of available periods restricted by the OSEK/VDX OS is ϕ. As briefly discussed in Section 1, control applications have to be sampled with one period or a combination of multiple periods from ϕ. In the latter case, switching between two sampling periods can only occur at the common multiplier of them, as has been illustrated in Figure 2. Often, the optimal sampling period for a control application does not belong to the set ϕ. The simple and straightforward method used in practice is to select the largest sampling period in ϕ that is smaller than the optimal one. Taking the example in Table 1, assuming that the optimal sampling period is 7.5ms, then 5ms is chosen as the sampling period to be used. This results in a higher processor load, which is an important design aspect.

Denoting e_i to be the WCET of a control application C_i, if the uniform sampling period is h, the processor load for C_i is

$$L_i = \frac{e_i}{h}. \tag{15}$$

The upper bound on the load of any processor is denoted as U. Considering a single processor p,

$$\sum_{\{i|C_i \text{ runs on } p\}} L_i \leq U. \tag{16}$$

Under the EDF scheduling, the upper bound U is equal to 1. Under the rate monotonic scheduling, U is equal to $m(2^{1/m} - 1)$, where m is the number of applications running on p (Liu and Layland 1973). A variety of tools, such as Inchron (2017), Timing Architects (2017), and Symtavision (2017), are used in the industry for more general scheduling analysis. Clearly, increasing the sampling period of a control application decreases its processor load and thus potentially enables more applications to be integrated on the ECU.

4 CONTROLLER WITH NON-UNIFORM SAMPLING

The design problem for a control application C_i in this work is to reduce the processor load L_i, while satisfying the settling time requirement $t^0_{s,i}$, the system stability and the input saturation constraint $U_{max,i}$. Towards this, we propose a controller with non-uniform sampling switching among multiple sampling periods in ϕ.

The cyclic sequence of sampling periods for a control application defines a schedule S,

$$S = \{T_1, T_2, T_3, \ldots, T_N\}, \tag{17}$$

where $\forall j \in \{1, 2, \ldots, N\},\ T_j \in \phi$. It implies the sequence of sampling periods as

$$T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_N \rightarrow T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_N \rightarrow repeat$$

Following the assumption in Equation (15) that the WCET of C_i is e_i, the processor load for C_i over S is

$$L_i = \frac{Ne_i}{\sum_{j=1}^{N} T_j}. \tag{18}$$

Dictated by the schedule S, N systems switch cyclically in a deterministic fashion. The dynamics of N systems within one cycle of S is (referring to Equation (14))

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{A}_d(T_1)\mathbf{x}[k] + \mathbf{B}_d(T_1)u[k-1], \\ \mathbf{x}[k+2] &= \mathbf{A}_d(T_2)\mathbf{x}[k+1] + \mathbf{B}_d(T_2)u[k], \\ &\vdots \\ \mathbf{x}[k+N] &= \mathbf{A}_d(T_N)\mathbf{x}[k+N-1] + \mathbf{B}_d(T_N)u[k+N-2]. \end{aligned} \tag{19}$$

The schedule $S^0 = \{$2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms$\}$ with seven systems is used as an example for the illustration purpose and shown in Figure 5. As discussed in Section 3.2, the actuation occurs at the end of a sampling period and the sensor-to-actuator delay is equal to one sampling period. For example, the control input $u[k]$ is computed based on the system state $x[k]$ at t_k and actuated at t_{k+1}.

We introduce a new augmented state $\mathbf{z}[k] = [\,\mathbf{x}[k]\ u[k-1]\,]^T$. Then, $\forall j \in \{1, 2, \ldots, N\}$,

$$\mathbf{z}[k+j] = \begin{bmatrix} \mathbf{A}_d(T_j) & \mathbf{B}_d(T_j) \\ \mathbf{0} & 0 \end{bmatrix} \mathbf{z}[k+j-1] + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} u[k+j-1], \tag{20}$$

where $\mathbf{0}$ is a zero vector. $\mathbf{A}_{aug}$ and $\mathbf{B}_{aug}$ are the system matrix and input matrix for the new augmented state, and denoted as

$$\mathbf{A}_{aug}(T_j) = \begin{bmatrix} \mathbf{A}_d(T_j) & \mathbf{B}_d(T_j) \\ \mathbf{0} & 0 \end{bmatrix}, \ \mathbf{B}_{aug}(T_j) = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}. \tag{21}$$

The system output is

$$y[k+j-1] = \mathbf{C}_{aug}\mathbf{z}[k+j-1], \tag{22}$$

where

$$\mathbf{C}_{aug} = [\,\mathbf{C}\ 0\,]. \tag{23}$$

The control input is designed as

$$u[k+j-1] = \mathbf{K}_j\mathbf{z}[k+j-1] + F_j r. \tag{24}$$

As shown in Figure 5, for example, $u[k]$ is computed from $x[k]$, $u[k-1]$, and K_1. Combining Equations (20), (21), and (24), the closed-loop dynamics is

$$\begin{aligned} \mathbf{z}[k+j] &= \mathbf{A}_{aug}(T_j)\mathbf{z}[k+j-1] + \mathbf{B}_{aug}(T_j)u[k+j-1] \\ &= (\mathbf{A}_{aug}(T_j) + \mathbf{B}_{aug}(T_j)\mathbf{K}_j)\mathbf{z}[k+j-1] + \mathbf{B}_{aug}(T_j)F_j r. \end{aligned} \tag{25}$$

We denote the closed-loop system matrix as

$$\mathbf{A}_{cl,j} = \mathbf{A}_{aug}(T_j) + \mathbf{B}_{aug}(T_j)\mathbf{K}_j. \tag{26}$$

It is noted that Equations (22), (24), (21), (25), and (26) are applied for every j in $\{1, 2, \ldots, N\}$. In the following of this section, we continue not to repeat the condition.

According to Equation (25), the overall dynamics within a cycle of the example S^0 is

$$\begin{aligned} \mathbf{z}[k+7] =& \mathbf{A}_{cl,7}\mathbf{z}[k+6] + \mathbf{B}_{aug}(T_7 = 5\text{ms})F_7 r \\ =& \mathbf{A}_{cl,7}(\mathbf{A}_{cl,6}\mathbf{z}[k+5] + \mathbf{B}_{aug}(T_6 = 5\text{ms})F_6 r) + \mathbf{B}_{aug}(T_7 = 5\text{ms})F_7 r \\ =& \mathbf{A}_{cl,7}\mathbf{A}_{cl,6}\mathbf{z}[k+5] + \mathbf{A}_{cl,7}\mathbf{B}_{aug}(T_6 = 5\text{ms})F_6 r + \mathbf{B}_{aug}(T_7 = 5\text{ms})F_7 r \\ =& \mathbf{A}_{cl,7}\mathbf{A}_{cl,6}(\mathbf{A}_{cl,5}\mathbf{z}[k+4] + \mathbf{B}_{aug}(T_5 = 2\text{ms})F_5 r) \\ & + \mathbf{A}_{cl,7}\mathbf{B}_{aug}(T_6 = 5\text{ms})F_6 r + \mathbf{B}_{aug}(T_7 = 5\text{ms})F_7 r \\ =& \mathbf{A}_{cl,7}\mathbf{A}_{cl,6}\mathbf{A}_{cl,5}\mathbf{z}[k+4] + \mathbf{A}_{cl,7}\mathbf{A}_{cl,6}\mathbf{B}_{aug}(T_5 = 2\text{ms})F_5 r \\ & + \mathbf{A}_{cl,7}\mathbf{B}_{aug}(T_6 = 5\text{ms})F_6 r + \mathbf{B}_{aug}(T_7 = 5\text{ms})F_7 r \\ & \vdots \\ =& \prod_{j=1}^{7}\mathbf{A}_{cl,j}\mathbf{z}[k] + \prod_{j=2}^{7}\mathbf{A}_{cl,j}\mathbf{B}_{aug}(2\text{ms})F_1 r + \prod_{j=3}^{7}\mathbf{A}_{cl,j}\mathbf{B}_{aug}(2\text{ms})F_2 r \\ & + \prod_{j=4}^{7}\mathbf{A}_{cl,j}\mathbf{B}_{aug}(2\text{ms})F_3 r + \prod_{j=5}^{7}\mathbf{A}_{cl,j}\mathbf{B}_{aug}(2\text{ms})F_4 r \\ & + \prod_{j=6}^{7}\mathbf{A}_{cl,j}\mathbf{B}_{aug}(2\text{ms})F_5 r + \mathbf{A}_{cl,7}\mathbf{B}_{aug}(5\text{ms})F_6 r + \mathbf{B}_{aug}(5\text{ms})F_7 r. \end{aligned} \tag{27}$$

If the pair $(\mathbf{A}_{aug}(T_j), \mathbf{B}_{aug}(T_j))$ is controllable, then the feedback gain $\mathbf{K}_j$ can be designed by pole-placement and computed as per Equation (12),

$$\mathbf{K}_j = [0 \ \ \cdots \ \ 0 \ \ 1]\, CO_j^{-1}\, \gamma_c(\mathbf{A}_{aug}(T_j)), \tag{28}$$

where

$$CO_j = [\mathbf{B}_{aug}(T_j) \quad \mathbf{A}_{aug}(T_j)\mathbf{B}_{aug}(T_j) \quad \cdots \quad \mathbf{A}_{aug}(T_j)^{n-1}\mathbf{B}_{aug}(T_j)]. \tag{29}$$

Poles to place are eigenvalues of $\mathbf{A}_{cl,j}$. The number of poles is $(n+1)N$. To ensure stability, eigenvalues of the overall closed-loop system matrix $\prod_{j=1}^{7}\mathbf{A}_{cl,j}$ must have absolute values of less than unity. The technique for pole-placement is introduced in the next section. The feedforward gain

F_j is computed in a similar way to Equation (13). In Equation (25), we let $\mathbf{z}[k+j] = \mathbf{z}[k+j-1]$ and $y[k+j-1] = \mathbf{C}_{aug}\mathbf{z}[k+j-1] = r$. Then we have

$$\mathbf{z}[k+j-1] = (\mathbf{A}_{aug}(T_j) + \mathbf{B}_{aug}(T_j)\mathbf{K}_j)\mathbf{z}[k+j-1] + \mathbf{B}_{aug}(T_j)F_j\mathbf{C}_{aug}\mathbf{z}[k+j-1]. \tag{30}$$

This equation is valid, no matter what value $\mathbf{z}[k+j-1]$ takes. Therefore,

$$\mathbf{I} = \mathbf{A}_{aug}(T_j) + \mathbf{B}_{aug}(T_j)\mathbf{K}_j + \mathbf{B}_{aug}(T_j)F_j\mathbf{C}_{aug}. \tag{31}$$

Then,

$$F_j = \frac{1}{\mathbf{C}_{aug}(\mathbf{I} - \mathbf{A}_{aug}(T_j) - \mathbf{B}_{aug}(T_j)\mathbf{K}_j)^{-1}\mathbf{B}_{aug}(T_j)}. \tag{32}$$

5 PSO-BASED POLE-PLACEMENT

We now formulate an optimization problem for the pole-placement as

$$\min_{\mathbb{D}} t_s \quad \text{subject to} \quad |u[k]| \le U_{max}, \quad t_s \le t_s^0, \tag{33}$$

where poles are decision variables. The settling time t_s, which can be evaluated with simulation depending on the decision variables, is to be minimized as the objective. There are three constraints. First, the input saturation has to be respected. Second, the settling time requirement has to be satisfied. Third, $\mathbb{D}$ is a domain of poles ensuring the stability of the overall system. We try to optimize the settling time beyond the requirement, so that the control performance can be maximized while nothing else (e.g., the processor load) needs to be compromised.

It is challenging to solve such a constrained non-convex optimization problem with significant non-linearity. We use the efficient PSO technique (Sedighizadeh and Masehian 2009). A group of particles are randomly initialized in the decision space with positions and velocities. The particles represent decision variables (i.e., controller poles). They search for the optimum by iteratively updating their positions. The search is led by two points. The first is the local best point that has been reached by a particle. Every particle has its own local best point. The second is the global best point that has been reached considering all particles. We let feasibility dominate performance when comparing two points:

- A point respecting all constraints is better than a point violating one or more constraints. That is, the objective value has no influence.
- If both points respect all constraints, then the point with a shorter settling time (i.e., the optimization objective) is considered better.
- If neither of the points respects all constraints (i.e., both of them violate at least one constraint), then still the point with a shorter settling time is considered better.

The velocity of a particle is determined by the following equation:

$$\mathbf{V}_{\text{new}} = \alpha_0\mathbf{V}_{\text{current}} + \alpha_1\text{rand}(0,1)(\mathbf{P}_{\text{lbest}} - \mathbf{P}_{\text{current}}) + \alpha_2\text{rand}(0,1)(\mathbf{P}_{\text{gbest}} - \mathbf{P}_{\text{current}}), \tag{34}$$

where $\mathbf{V}_{\text{new}}$ is the new velocity, $\mathbf{V}_{\text{current}}$ is the current velocity, $\mathbf{P}_{\text{current}}$ is the current position, $\mathbf{P}_{\text{lbest}}$ is the local best point of this particle and $\mathbf{P}_{\text{gbest}}$ is the best point of all particles. $\text{rand}(0,1)$ is a random number with uniform distribution from the open interval $(0,1)$. α_0 is the weight inertia. α_1 and α_2 are cognitive and social scaling parameters. Widely used values for these parameters are

$$\alpha_0 = 0.4, \quad \alpha_1 = \alpha_2 = 2, \tag{35}$$

which have been shown to have good performance in many optimization scenarios. The new position of this particle is

$$\mathbf{P}_{\text{new}} = \mathbf{P}_{\text{current}} + \mathbf{V}_{\text{new}}. \tag{36}$$

The algorithm is terminated once all particles have converged or the maximum number of iterations has been reached. The timing complexity of PSO is clearly polynomial.

ALGORITHM 1: Pole-placement with PSO

```
Input: $PoleNum$, $ParticleNum$, $IterationNum$
Output: $\mathbf{P}_{\text{gbest}}$
1  for $i \leftarrow 1$ to $ParticleNum$ do
2      for $j \leftarrow 1$ to $PoleNum$ do
3          Randomly initialize $P^i_j$ in $[0, 1]$
4          $V^i_j = 0$
5      end
6      $\mathbf{P}^i_{\text{lbest}} = \mathbf{P}^i = \{P^i_1, P^i_2, \ldots, P^i_{PoleNum}\}$
7  end
8  Record $\mathbf{P}_{\text{gbest}}$
9  $k = 0$
10 while $k < IterationNum$ and not all particles have converged do
11     Update $V^i_j$ and $P^i_j$ with (34) and (36)
12     for $i \leftarrow 1$ to $ParticleNum$ do
13         Update $\mathbf{P}^i_{\text{lbest}}$
14     end
15     Update $\mathbf{P}_{\text{gbest}}$
16     $k = k + 1$
17 end
```

The pseudocode is shown in Algorithm 1 to illustrate the pole-placement with PSO. Every pole of every particle is randomly initialized in $[0, 1]$ (Line 3). This gives the initilized particles a good chance of being feasible (i.e., satisfying all the constraints). The velocity is initialized to be 0 (Line 4). The local best point of every particle (Line 6) and the global best point of all particles (Line 8) are recorded. Afterwards, we iteratively update the position and velocity of every particle (Line 11). At every iteration, we record the local best point of each particle (Line 13) and the global best point of all particles (Line 15). Once the algorithm is terminated, the global best point is returned. It is noted that we do not impose hard feasibility requirement in this algorithm. The rules that prioritize feasibility over performance when comparing two points drive the search towards the feasible region. The final solution has the best performance among all the visited feasible points during the search.

One major issue with PSO is its tendency for fast and premature convergence before the global optimum has been found, since its search is highly directional (Sedighizadeh and Masehian 2009). This problem gets more severe as the number of dimensions in the decision space grows larger. The cognitive and social scaling parameters α_1 and α_2 have a significant impact on the search behavior and convergence of PSO. If α_1 is larger than α_2, then the PSO tends to have better local searches, yet converges more slowly. If α_2 is larger than α_1, then the PSO often converges fast before thoroughly searching the local area around each visited point, thereby possibly missing the global optimum. This is a tradeoff between optimality and efficiency. It is challenging to achieve both simultaneously.

There have been a number of works extensively investigating the parameterization of PSO (Jordehi and Jasni 2013; Pedersen 2010; Nickabadi et al. 2011). Various existing strategies for PSO parameters setting are summarized in Jordehi and Jasni (2013), which discusses some future research directions. A list of good parameter choices for several benchmarks is reported in Pedersen (2010). An adaptive inertia weight is proposed in Nickabadi et al. (2011) and uses the success rate of the swarm as its feedback parameter to ascertain the particles' situation in the search space. In this work, we propose an adaptive parameterization approach for the cognitive and social scaling parameters with a constant sum. As the iteration number increases, α_1 is decreased and α_2 increases. The basic idea is that at the beginning of the optimization when particles are more disperse, local areas are better searched aiming to explore a larger space. When the optimization approaches to the end, particles are close to one another, and the focus is placed on convergence. The goal is to achieve optimality and efficiency at the same time.

Assuming that the iteration number is q ($0 < q \leq q_{max}$, where q_{max} is the maximum number of iterations), the cognitive and social scaling parameters can be computed as

$$\alpha_2 = f\left(\frac{q}{q_{max}}\right), \quad \alpha_1 = 4 - \alpha_2, \tag{37}$$

where the constant sum of α_1 and α_2 is taken as 4. f is a function that can be customarily decided. In this work, we use an exponential function as

$$f(x) = 0.5e^{2x} + 0.1. \tag{38}$$

A numerical example is used to show the advantage of the proposed adaptively parameterized PSO technique. The formulation is as follows:

$$\begin{gathered}\max_{\mathbb{D}} \varphi = e^{-\frac{1}{3}{\beta_1}^3 + \beta_1 - {\beta_2}^2} \\ \text{subject to} \\ \mathbb{D} = \{(\beta_1, \beta_2) | -1.8 \leq \beta_1 \leq 2, \ -2 \leq \beta_2 \leq 2\},\end{gathered} \tag{39}$$

where β_1 and β_2 are two continuous decision variables, constrained in the decision space $\mathbb{D}$. The objective to maximize is φ. The conventional PSO method is illustrated in Figure 6. Five particles are randomly initialized at p_1, p_2, p_3, p_4, and p_5, as shown in Table 2. Among them, p_1 has the best objective value. After 13 iterations, all five particles converge to points around the local optimum p_l ($\beta_1 = -1.7997, \beta_2 = 8.3188 \times 10^{-3}, \varphi = 1.1540$). The path showing how the global best point evolves iteratively is drawn, with certain points that are too close to others omitted for better illustration. It can be seen that the search is highly directed towards the global best point in each iteration. The global optimum is not found and particles quickly converge to the local optimum before exploring the decision space sufficiently.

The proposed novel PSO method with adaptive parameterization is illustrated in Figure 7. The same initial points are used as in Figure 6 and Table 2. After 13 iterations, all five particles converge to points around the global optimum p_g ($\beta_1 = 9.8765 \times 10^{-1}, \beta_2 = -3.9853 \times 10^{-3}, \varphi = 1.9474$). The path showing how the global best point evolves iteratively is drawn, with certain points too close to others omitted. It can be seen that the decision space is better explored and the global optimum is found. This numerical example shows the advantage of the novel PSO method over the conventional one. Despite the good performance, we remark that our proposed optimization technique is a heuristic and does not guarantee global optimality.

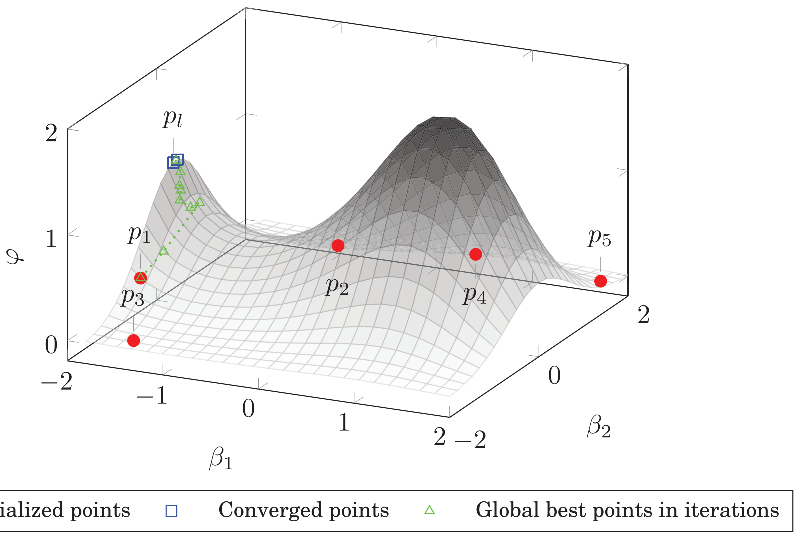

Fig. 6. With the conventional particle swarm optimization method, five particles are randomly initilized and converge to the local optimum p_l.

Table 2. Randomly Initialized Particles in the Numerical Example of PSO

Particle	p_1	p_2	p_3	p_4	p_5
β_1	−1.7	−0.8	−1.4	0.5	1.9
β_2	−1	1.5	−1.8	1.8	1.6
φ	0.3456	0.0562	0.0241	0.0619	0.0525

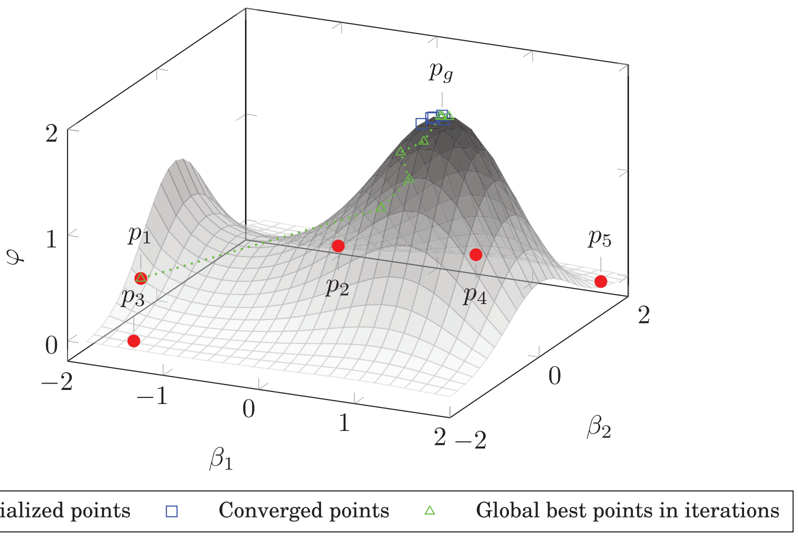

Fig. 7. With the proposed novel particle swarm optimization method with adaptive parameterization, five particles are randomly initialized and converge to the global optimum p_g.

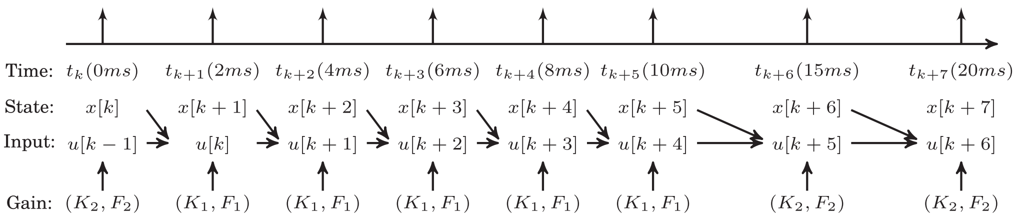
Time: $t_k(0ms)$ $t_{k+1}(2ms)$ $t_{k+2}(4ms)$ $t_{k+3}(6ms)$ $t_{k+4}(8ms)$ $t_{k+5}(10ms)$ $t_{k+6}(15ms)$ $t_{k+7}(20ms)$
State: $x[k]$ $x[k+1]$ $x[k+2]$ $x[k+3]$ $x[k+4]$ $x[k+5]$ $x[k+6]$ $x[k+7]$
Input: $u[k-1]$ $u[k]$ $u[k+1]$ $u[k+2]$ $u[k+3]$ $u[k+4]$ $u[k+5]$ $u[k+6]$
Gain: (K_2, F_2) (K_1, F_1) (K_1, F_1) (K_1, F_1) (K_1, F_1) (K_1, F_1) (K_2, F_2) (K_2, F_2)

Fig. 8. Illustration of the alternative more scalable controller design with an example schedule S^0. Gains are the same for systems with the same sampling period. The figure is not drawn to scale.

6 ALTERNATIVE CONTROLLER DESIGN FOR SCALABILITY

As discussed before, the number of dimensions in the decision space using the controller design presented in Section 4 is $(n + 1)N$. When the number of sampling periods N in a schedule is very large, solving the pole-placement optimization problem could be computationally too heavy, even for an offline task. The PSO-based technique naturally offers a solution—decreasing the number of particles and iterations. However, this renders the result stochastic with a large variation and considerably dependent on the choices during initialization, which is often undesirable. In this section, we provide an alternative controller design aiming for better scalability on the number of sampling periods.

The complexity of the proposed controller in Section 4 comes from that the closed-loop dynamics of all the sampling periods are considered and optimized. A simpler design technique is to assume identical closed-loop dynamics for the systems with the same sampling period (the same open-loop dynamics as well). That is, if $T_j = T_{j'}$, the poles and feedback/feedforward gains of these two systems are the same. Taking S^0 as an example, as shown in Figure 8, for the five systems with the sampling period of 2ms, poles are assumed to be the same. Therefore, the feedback gains are all $\mathbf{K}_1$ and the feedforward gains are all F_1. The closed-loop system matrix $\mathbf{A}_{aug}(2\text{ms}) + \mathbf{B}_{aug}(2\text{ms})\mathbf{K}_1$ is considered in the pole-placement. Similarly, for the two systems with the sampling period of 5ms, feedback gains are $\mathbf{K}_2$ and feedforward gains are F_2. The closed-loop system matrix $\mathbf{A}_{aug}(5\text{ms}) + \mathbf{B}_{aug}(5\text{ms})\mathbf{K}_2$ is considered in the pole-placement. Everything else remains unchanged with respect to the design described in Section 4 and Section 5.

Clearly, the solution is suboptimal, since the assumption that poles are identical for systems with the same sampling period does not necessarily hold. The advantage is a smaller decision space. The number of decision variables (i.e., poles to place) becomes $(n + 1)N'$—the number of states of the plant multiplied by the number of distinctive sampling periods in the schedule. For the example schedule S^0, N' is 2 and thus the number of dimensions in the decision space is $\frac{2}{7}$ of the one in Section 4. Therefore, when the number of sampling periods N in the schedule S is very large and the number of distinctive sampling periods N' is relatively small, this alternative controller has better scalability on the number of sampling periods.

7 EXPERIMENTAL RESULTS

System description: The proposed OS-aware controller design technique is evaluated on an experimental EMB system from Bosch used in automobiles with simulation. The simplified model is shown in Figure 9. When the EMB is active, the braking caliper should reach a reference position r, which is at the braking disc, within the desired settling time t_s^0. This is the *position* mode. The requirement on the settling time ensures the reactiveness of the system. After that, a certain force is applied in the *force* mode. The electric motor mobilizing the braking caliper is powered by the onboard battery, which has a voltage of 12V. In this work, we consider the position mode

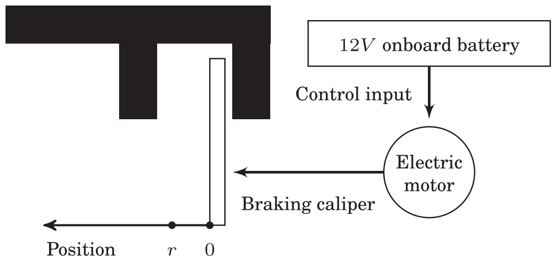

Fig. 9. A simplified model of the electro-mechanical braking system.

Table 3. EMB System Requirements

Settling time requirement	Input saturation	Reference position	WCET
150ms	12V	2mm	0.7ms

Table 4. Settling Time and Processor Load of Three Schedules

Schedule	Settling time	Requirement satisfaction	Processor load
$S1 = \{5\text{ms}\}$	256.40ms	Violated	14%
$S2 = \{2\text{ms}\}$	113.27ms	Satisfied	35%
S^0 (novel PSO)	132.14ms	Satisfied	24.5%
S^0 (conventional PSO)	154.05ms	Violated	24.5%

of the EMB system, which is of interest in several scenarios: braking, disk wiping, and pre-crash preparations. The system dynamics can be modeled as

$$A = \begin{bmatrix} -520 & -220 & 0 & 0 & 0 \\ 220 & -500 & -999994 & 0 & 2 \times 10^8 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 66667 & -0.1667 & -1.3333 \times 10^7 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{40}$$
$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

There are five system states—motor current, motor angular velocity, motor angular position, caliper velocity, and caliper position. The control input is the applied voltage on the motor. The requirements are summarized in Table 3. The set of available sampling periods offered by OSEK/VDX OS is

$$\phi = \{1\text{ms}, 2\text{ms}, 5\text{ms}, 10\text{ms}, 20\text{ms}, 50\text{ms}, 100\text{ms}, 200\text{ms}, 500\text{ms}, 1\text{s}\}. \tag{41}$$

In the experiments of this work, we consider the EDF scheduling under the OSEK/VDX OS. When the deadlines are the same, the application with a smaller index has a higher priority.

PSO-based controller design with non-uniform sampling: As shown in Table 4 and Figure 10, the schedule $S1 = \{5\text{ms}\}$ cannot meet the settling time requirement. The largest sampling period smaller than 5ms in ϕ is 2ms. The schedule $S2 = \{2\text{ms}\}$ is able to fulfill all the requirements. According to Equation (15), using the WCET requirement in Table 3, the processor load of $S2$ is 35%. As discussed before, this number can be unnecessarily large and prevents more applications from sharing the ECU.

Then we evaluate the schedule $S^0 = \{2\text{ms}, 2\text{ms}, 2\text{ms}, 2\text{ms}, 2\text{ms}, 5\text{ms}, 5\text{ms}\}$ switching between 2ms and 5ms. This sequence of sampling periods satisfies the OSEK/VDX OS requirement as

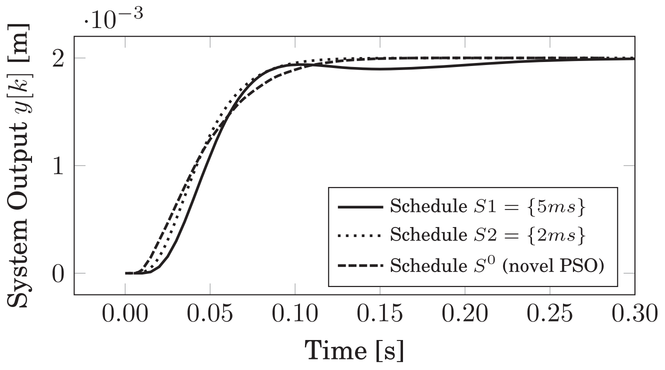

Fig. 10. System output of three different schedules. The proposed PSO technique is used for S^0.

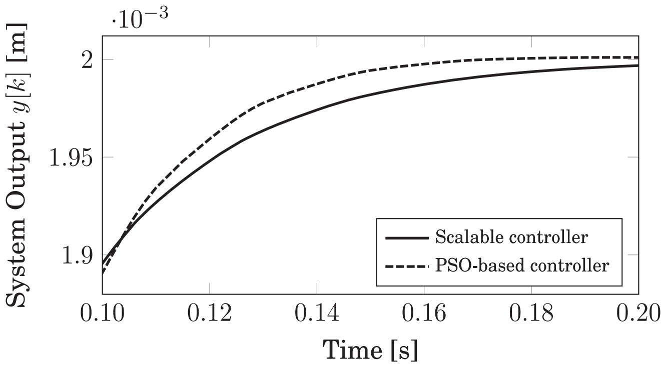

Fig. 11. System output of the PSO-based controller and its scalable variant.

Table 5. Comparison of the PSO-based Controller Design with Its Scalable Variant

Controller	Settling time	Requirement satisfaction	Particle	Iteration	Time
PSO-based	132.14ms	Satisfied	50	16	1448s
Scalable	147.07ms	Satisfied	15	6	28s

discussed in Section 3.2. The controller with non-uniform sampling is designed as proposed in Section 4 and the novel PSO with adaptive parameterization as in Section 5 is used for pole-placement. 50 particles are deployed and converge after 16 iterations. Increasing the number of particles beyond 50 does not further improve the control performance. There are 42 poles from the seven closed-loop system matrices. S^0 has a slightly longer settling time than $S2$, yet still fulfills the requirement. According to Equation (18), the processor load is 24.5%, achieving a 30% reduction compared to $S2$. We also evaluate the settling time of S^0 using the conventional PSO technique. Fifty particles are used, and the convergence also takes 16 iterations. As reported in Table 4, the solution does not satisfy the requirement.

Scalable design: Comparison of the PSO-based controller presented in Section 4 and its scalable variant in Section 6 is shown in Figure 11 and Table 5. We zoom to the region around the settling of the system output, where the two plots are very close to each other. The settling time generated

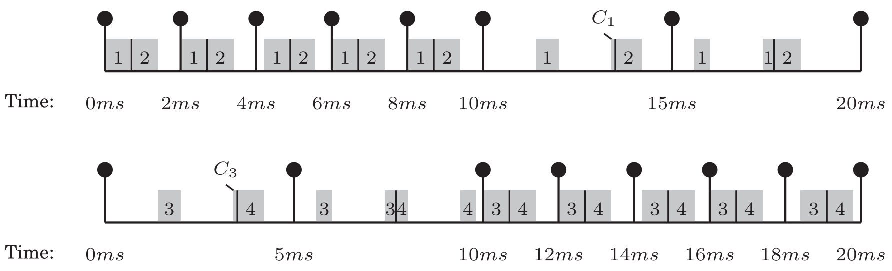

Fig. 12. Invocation timing of four control applications under the schedule S^0. The schedule for C_1 and C_2 is {2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms}, and the schedule for C_3 and C_4 is {5ms, 5ms, 2ms, 2ms, 2ms, 2ms, 2ms}. Numbers 1, 2, 3, and 4 represent the applications C_1, C_2, C_3, and C_4, respectively. Preemption is allowed in OSEK/VDX OS.

Table 6. Exact Invocation Starting Times of Four Control Applications Under the Non-uniform Sampling Schedule S^0

Invocation number	C_1	C_2	C_3	C_4
1	0	0.7	1.4(0.6)/3.4(0.1)	3.5
2	2	2.7	5.6(0.4)/7.4(0.3)	7.7(0.3)/9.4(0.4)
3	4.2	4.9	10	10.7
4	6	6.7	12	12.7
5	8	8.7	14.2	14.9
6	11.4(0.6)/13.4(0.1)	13.5	16	16.7
7	15.6(0.4)/17.4(0.3)	17.7	18.4	19.1

When one invocation is preempted, two starting times are separated by a forward slash, and the number in the bracket indicates the duration. The timing unit is ms.

by the scalable controller is longer, yet still satisfies the requirement. It takes 15 particles that converge after 6 iterations. Increasing the number of particles beyond 15 does not further improve the control performance. The total computation time on a computer with an Intel i5 processor operating at 2.6GHz with 4GB RAM is 28s, compared to 1448s for the controller aiming for optimality. Although 1448s sounds acceptable for an offline task, when a schedule has more sampling periods than S^0, the computation could take hours or even days due to the increase of the decision space dimensions. In such a case, if the number of distinctive sampling periods is small, the scalable controller design is preferred.

Packing of more applications: Now we consider a case that multiple applications are to be implemented on ECUs. For the convenience of illustration, all applications are assumed to be identical to the EMB system discussed before with the WCET of 0.7ms. As reported above, the schedule $S2 = \{2ms\}$ is able to satisfy the control performance requirement and system constraints. Under $S2$, an ECU is able to accommodate two applications according to Equation (16). Under the non-uniform sampling schedule S^0, four applications can share one ECU, where detailed invocation timing is presented in Figure 12 and Table 6. Numbers 1, 2, 3, and 4 represent the applications C_1, C_2, C_3, and C_4, respectively. While the schedule for C_1 and C_2 is {2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms}, the schedule for C_3 and C_4 is {5ms, 5ms, 2ms, 2ms, 2ms, 2ms, 2ms}. The switching of sampling periods occurs every 10ms for both schedules. When an application is in the shorter sampling period (2ms in this case), it occupies the ECU more often. Therefore, these two variants of S^0 are essentially opposite to each other, so that when some applications require more frequent access to the

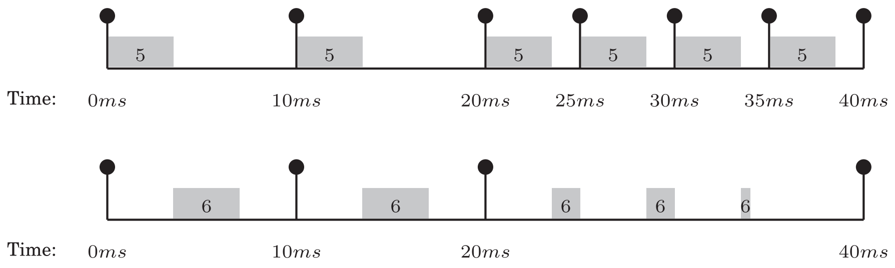

Fig. 13. Invocation timing of C_5 and C_6 under the non-uniform sampling schedules. The schedule for C_5 is {10ms, 10ms, 5ms, 5ms, 5ms, 5ms}, and the schedule for C_6 is {10ms, 10ms, 20ms}. Numbers 5 and 6 represent the applications C_5 and C_6, respectively.

Table 7. Exact Invocation Starting Times of C_5 and C_6 under the Non-uniform Sampling Schedules

Invocation number	1	2	3	4	5	6
C_5	0	10	20	25	30	35
C_6	3.5	13.5	23.5(1.5)/28.5(1.5)/33.5(0.5)	N.A.	N.A.	N.A.

When one invocation is preempted, two starting times are separated by a forward slash and the number in the bracket indicates the duration. The timing unit is ms.

ECU, others are in the longer sampling period (5ms in this case), requesting the execution less often. In this way, the number of applications packed onto the ECU can be maximized.

It is noted that preemption is supported in OSEK/VDX OS. In this case, we consider the EDF scheduling. For instance, at 1.4ms when both C_1 and C_2 finish their first invocations, C_3 is started and allowed to access the processor for 0.6ms. After that, C_3 is suspended, waiting for the second invocations of C_1 and C_2, which have higher priorities. Then, C_3 resumes and completes its first invocation.

It can be seen that the number of applications that are accommodated by an ECU is doubled with the proposed OS-aware non-uniform sampling controller design, which is significant improvement for the cost-sensitive automotive domain.

To further validate the advantages brought by the proposed approach, we consider another case consisting of applications with different non-uniform sampling schedules. We assume that the schedule with the uniform sampling period 5ms is able to satisfy the performance requirement of the application C_5. We also assume that the non-uniform sampling schedule with two sampling periods 5ms and 10ms satisfies the performance requirement of C_5. However, the schedule with the uniform sampling period 10ms does not satisfy the performance requirement of C_5. The performance requirement of the application C_6 can be satisfied with the schedule {10ms} and {10ms, 10ms, 20ms}, yet not {20ms}. The WCETs of both C_5 C_6 are 3.5ms. If only uniform sampling schedules are considered, then C_5 and C_6 cannot be implemented on one processor, since the total processor load is $3.5/5 + 3.5/10 = 1.05$, which exceeds the upper bound 1 as discussed in Equation (16). If non-uniform sampling schedules are deployed, then both C_5 and C_6 can be implemented on one processor. Detailed invocation timing is presented in Figure 13 and Table 7. It can be seen that the processor can run another strictly periodic application. If the period is 5ms, then the longest allowed WCET is 0.625ms. It is noted that the third invocation of C_6 will be evenly distributed into the third to the sixth invocation of C_5.

8 CONCLUDING REMARKS

To deal with the restriction imposed by the OS on sampling periods for control applications, we present a novel performance-oriented controller design with a non-uniform sampling schedule, in which an adaptively parameterized PSO is used for pole-placement. It reduces the processor load, while satisfying the control performance requirement and system constraints. This saves computation resources and enables integration of more functions and applications into an ECU, thereby saving costs, which is critical in the automotive domain. Experimental results show that the number of control applications sharing an ECU can be higher with the proposed OS-aware controller design with non-uniform sampling.

The focus of this article is to show that more applications can be packed using a non-uniform sampling schedule and the proposed controller design method. A relevant question for the future works is the design of the optimal sampling schedule. Given that the controller design has to be redone for every possible schedule, the optimal schedule design is a non-trivial problem, especially when the number of applications sharing one processor is large. However, by checking the performance of uniform sampling schedules (e.g., the period of 2ms is able to satisfy the performance requirement and the period of 5ms is not), it is possible to intuitively come up with a few non-uniform sampling schedules as candidates of design interest (e.g., schedules switching between 2ms and 5ms, 1ms, and 10ms, etc.) to be evaluated with the approach proposed in this article.

While in this article the focus was on single-core ECUs, we intend to extend our approach to multi-core architectures. There are mainly two challenges to address. First, due to load balancing requirements, it might be necessary to distribute different parts of complex control applications to different cores. This introduces additional delays for sensor-to-actuator cause-effect chains that need to be taken into account during controller design to ensure stability. Second, memory partitioning and code placement need to be considered, since they have a major influence on the execution times of control programs.

REFERENCES

2017. Inchron GmbH. Retrieved from https://www.inchron.de/.

2017. Symtavision GmbH. Retrieved from https://www.symtavision.com/.

2017. Timing Architects. Retrieved from http://www.timing-architects.com/.

Juergen Ackermann and Vadim Utkin. 1998. Sliding mode control design based on Ackermann's formula. *IEEE Trans. Automat. Control* 43, 2 (1998), 234–237.

Adolfo Anta and Paulo Tabuada. 2009. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS'09).*

Vincenzo Apuzzo, Alessandro Biondi, and Giorgio C. Buttazzo. 2016. OSEK-like kernel support for engine control applications under EDF scheduling. In *Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'16).*

Karl J. Åström and Richard M. Murray. 2009. *Feedback Systems: An Introduction for Scientists and Engineers.* Princeton University Press.

Enrico Bini and Giuseppe M. Buttazzo. 2014. The optimal sampling pattern for linear control systems. *IEEE Trans. Automat. Control* 59, 1 (2014), 78–90.

Giorgio Buttazzo and Paolo Gai. 2006. Efficient EDF implementation for small embedded systems. In *Proceedings of the 2006 International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'06).*

Rosa Castane, Pau Marti, Manel Velasco, Anton Cervin, and Dan Henriksson. 2006. Resource management for control tasks based on the transient dynamics of closed-loop systems. In *Proceedings of the 18th. Euromicro Conference on Real-Time Systems (ECRTS'06).*

Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. 2002. Feedback-feedforward scheduling of control tasks. *Real-Time Syst.* 23, 1–2 (2002), 25–53.

Anton Cervin, Manel Velasco, Pau Marti, and Antonio Camacho. 2011. Optimal online sampling period assignment: Theory and experiments. *IEEE Trans. Control Syst. Technol.* 19, 4 (2011), 902–910.

Xi Chen, Akramul Azim, Xue Liu, Sebastian Fischmeister, and Jun Ma. 2014. DTS: Dynamic TDMA scheduling for networked control systems. *J. Syst. Arch. Embed. Syst. Des.* 60, 2 (2014), 194–205.

OSEK/VDX Consortium. 2005. OSEK/VDX operating system specification Version 2.2.3.

Peter H. Feiler. 2003. *Real-time Application Development with OSEK: A Review of the OSEK Standards.* Technical Report. Carnegie Mellon University.

Luca Greco, Daniele Fontanelli, and Antonio Bicchi. 2011. Design and stability analysis for anytime control via stochastic scheduling. *IEEE Trans. Automat. Control* 56, 3 (2011), 571–585.

Shengyan Hong, Xiaobo Sharon Hu, Tao Gong, and Song Han. 2015. On-line data link layer scheduling in wireless networked control systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15).*

Shengyan Hong, Xiaobo Sharon Hu, and Michael Lemmon. 2010. Reducing delay jitter of real-time control tasks through adaptive deadline adjustments. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS'10).*

Kyusoo Jeong, Donggon Lee, Sungwook Park, and Changsik Lee. 2011. Effect of two-stage fuel injection parameters on NOx reduction characteristics in a DI diesel engine. *Energies* 4, 11 (2011), 2049–2060.

Ahmad Rezaee Jordehi and Jasronita Jasni. 2013. Parameter selection in particle swarm optimization: A survey. *J. Exp. Theor. Artif. Intell.* 25, 4 (2013), 527–542.

Simon Kramer, Dirk Ziegenbein, and Arne Hamann. 2015. Real world automotive benchmark for free. In *Proceedings of the 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'15).*

Eugene Lavretsky and Kevin Wise. 2013. *Robust and Adaptive Control with Aerospace Applications.* Springer.

Michael Lemmon and Xiaobo Sharon Hu. 2011. Almost sure stability of networked control systems under exponentially bounded bursts of dropouts. In *Proceedings of the 14th. ACM International Conference on Hybrid Systems: Computation and Control (HSCC'11).*

Hai Lin and Panos J. Antsaklis. 2009. Stability and stabilizability of switched linear systems: A survey of recent results. *IEEE Trans. Automat. Control* 54, 2 (2009), 308–322.

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

Ahmad Nickabadi, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. 2011. A novel particle swarm optimization algorithm with adaptive inertia weight. *Appl. Soft Comput.* 11, 4 (2011), 3658–3670.

Magnus Erik Hvass Pedersen. 2010. *Good Parameters for Particle Swarm Optimization.* Technical Report. Hvass Laboratories.

Dobrivoje Popovic, Mrdjan Jankovic, Steve Magner, and A. Teel. 2003. Extremum seeking methods for optimization of variable cam timing engine operation. In *Proceedings of the 2003 American Control Conference (ACC'03).*

James B. Rawlings and David Q. Mayne. 2009. *Model Predictive Control: Theory and Design.* Nob Hill Publishing.

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 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'16).*

Davoud Sedighizadeh and Ellips Masehian. 2009. Particle swarm optimization methods, taxonomy and applications. *Int. J. Comput. Theory Eng.* 1, 4 (2009), 486–502.

Dong Yue, Engang Tian, and Qinglong Han. 2013. A delay system method for designing event-triggered controllers of networked control systems. *IEEE Trans. Automat. Control* 58, 2 (Feb. 2013), 475–481.

Received September 2016; revised June 2017; accepted July 2017