Revisiting Dynamic Scheduling of Control Tasks: A Performance-Aware Fine-Grained Approach

Sunandan Adhikary, *Graduate Student Member, IEEE*, Ipsita Koley, Saurav Kumar Ghosh, Sumana Ghosh, and Soumyajit Dey, *Senior Member, IEEE*

***Abstract*—Modern cyber–physical systems (CPSs) employ an increasingly large number of software control loops to enhance their autonomous capabilities. Such large task sets and their dependencies may lead to deadline misses caused by platform-level timing uncertainties, resource contention, etc. To ensure the schedulability of the task set in the embedded platform in the presence of these uncertainties, there exist co-design techniques that assign task periodicities such that control costs are minimized. Another line of work exists that addresses the same platform schedulability issue by skipping a bounded number of control executions within a fixed number of control instances. Considering that control tasks are designed to perform robustly against delayed actuation (due to deadline misses, network packet drops etc.) a bounded number of control skips can be applied while ensuring certain performance margin. Our work combines these two control scheduling co-design disciplines and develops a strategy to adaptively employ control skips or update periodicities of the control tasks depending on their current performance requirements. For this we leverage a novel theory of automata-based control skip sequence generation while ensuring periodicity, safety and stability constraints. We demonstrate the effectiveness of this dynamic resource sharing approach in an automotive Hardware-in-loop setup with realistic control task set implementations.**

***Index Terms*—Adaptive scheduling, control system synthesis, cyber–physical systems (CPSs).**

I. Introduction

With the increasing number of autonomous features available in modern-day cyber–physical systems (CPSs), the corresponding task sets to be executed in their electronic control units (ECUs) have also increased significantly. To cope with this and utilize bandwidth efficiently, significant research has been reported in the domain of dynamic scheduling of control tasks, where unlike static schedules, depending on run-time performance requirements, the task period is dynamically switched while satisfying the stability and baseline performances [1]. Such techniques enable control tasks with steady system response to operate in lower frequencies. This, in turn, allows other control tasks that occasionally require more frequent actuation for maintaining the desirable performance to switch to a higher frequency, availing the relinquished bandwidth. However, such *multirate* approach of control task co-scheduling is limited by the allowable choice of sampling periods for each task and their joint schedulability for the task set. As a fine-grained alternative scheduling approach, researchers have proposed to convert hard temporal scheduling constraints for each job in a task into weakly hard ones [2]. *Weakly hard* constraints of a closed loop system is often captured as (m, k)-firm specifications [3] where a maximum of $(k - m)$ deadline misses or *control execution skips* are allowed in every k consecutive control task instances to maintain a desired performance. Such control skips are often introduced due to platform-level faults, jitters, task execution overruns, communication delays, etc., in embedded processing units and/or complex network components. Numerous research has been carried out over the past few years exploring systems performances under different weakly hard constraints [4], [5], [6], [7] and identifying weakly hard constraints that can be leveraged to co-schedule control tasks in a more efficient way [6], [8].

Related Work: Analysis and synthesis of control execution sequences with skips has been done in the literature with two primary objectives, a) establishing stability under various uncertainties for handling deadline overruns, b) leveraging control execution skips for the bandwidth-efficient co-design.

The first kind of work focuses on weakly hard modeling of control systems, analyzing their stability under timing uncertainties introduced due to faults or interference of other tasks [4], [5], [7], [9], [10]. For example, Pazzaglia et al. [4] analyzed the effect of deadline misses (caused by faults) on the stability of the closed-loop and its control cost. This is done by analyzing all possible deadline-miss sequences allowed by weakly hard specifications and applying novel scheduling (e.g., killing the current job that missed its deadline or letting it continue and killing the later jobs instead) and control (e.g., actuating with the last control input or resetting the control input) strategies to maintain asymptotic stability or minimize control cost. Works done by Linsenmayer and Allgower [5] and Pazzaglia et al. [10] looked into this problem from a switched system perspective, where the system under control actuation/execution and system under missed control actuation/execution due to packet drops or deadline-miss are

Manuscript received 3 August 2024; accepted 3 August 2024. This work was supported in part by the Qualcomm Innovation Fellowship. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Sunandan Adhikary.)*

Sunandan Adhikary, Ipsita Koley, Saurav Kumar Ghosh, and Soumyajit Dey are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, India (e-mail: mesunandan@kgpian.iitkgp.ac.in; ipsitakoley@iitkgp.ac.in; saurav.kumar.ghosh@cse.iitkgp.ac.in; soumya@cse.iitkgp.ac.in).

Sumana Ghosh is with the Computer and Communication Science Unit, Indian Statistical Institute, Kolkata 700108, India (e-mail: sumana@isical.ac.in).

Digital Object Identifier 10.1109/TCAD.2024.3443007

considered as two modes to switch between. They analyze and bound the maximum asymptotic growth of such a system under all possible control skipping strategies given as a weakly hard specification. In [9], this growth is quantified in terms of *joint spectral radius*. Pazzaglia et al. [11] computed and schedule stable controller gains to mitigate the effects of such deadline-miss scenarios on system stability.

The limitations of these works can be summarized in the following two points.

1) With fault-tolerance being the primary objective, such works often do not consider realistic performance criteria like *exponential decay rate* (explained later in Definition 1) as their objectives and mostly rely on analytical [4], [9] or heuristic-based [7] approaches for assuring asymptotic stability. Works that give theoretical stability guarantee mostly rely on common Lyapunov function (CLF)-based stability for a given performance criteria. This heavily restricts the number of possible control theoretic switching sequences which transpire in the platform as stable deadline-miss (i.e., control execution skip) possibilities [5], [12].
2) Moreover, most of the aforementioned works evaluate their strategies on different standalone control system models. How effectively these strategies scale and contribute to the overall bandwidth sharing between multiple closed loops in real-world networked, embedded control systems is not evaluated. Although works like [7] consider a realistic implementation setup, they rely on heuristic-based scheduling under faulty scenarios that lack theoretical underpinning.

The second kind of state-of-the-art (SOTA) works utilize these weakly hard specifications to bound intentionally skipped control executions and achieve better-scheduling solutions while ensuring performance or safety guarantees [6], [8], [13]. Xu et al. [8] achieved a resource-aware *aperiodic* schedule (different time intervals between multiple executions) by switching between all possible deadline-miss sequences that abide by a weakly hard constraint. They deploy a scheduler automaton that ensures state deviation within a safe upper bound, whereas works like [6], [13] address a similar problem by developing an automaton that skips control executions abiding by certain weakly hard constraints derived from a given performance criteria. Ghosh et al. [13] used a multirate controller gain scheduling approach to ensure performance during the deployment of such resource-optimized control execution schedules. Works like [1], [20], on the other hand, do not rely on weakly hard constraints but propose an optimal sampling period assignment method to achieve a similar optimized resource consumption objective. They assign optimal periodicities for the controllers by analyzing the cumulative control costs or system output deviations incurred by a schedulable set of periodicities. To avoid unstable behaviors and overshoot/undershoot beyond a safe range while switching between periodicities, most of these works again use asymptotic stability or CLF-based theoretical analysis. Similar criticisms (as described in the last paragraph) are applicable to these co-design-focused works as well, i.e., they rely on a CLF-based stability guarantee and do not consider a realistic implementation setup. However, the work in [14] aptly analyses the maximum number of allowable control execution skips in an embedded platform to ensure certain stability criteria, using multiple Lyapunov function (MLF)-based approach that is less restrictive than CLF-based approaches (more allowable skips). This also does not consider a realistic implementation in the presence of multiple closed loops.

Novelty and Contributions: Our main motivation stems from the limitations of these SOTA works. For a set of control loops and their performance criteria, our methodology, as highlighted in Fig. 1, performs a stepwise synthesis and deployment as discussed next.

1) For each participating control loop, we work out an MLF-based performance-aware switching strategy for switching between the choices of periodicities as well as the choices of control execution skips that operate within a given safe region (box 1 in Fig. 1). The use of exponential decay-based performance criteria makes our methodology more applicable to real-world system design, and the use of MLF admits more control-scheduling choices than SOTA works.
2) For each participating control loop, we synthesize a control skipping automaton (CSA), which utilizes the theoretically derived stable switching strategies to generate stable aperiodic activation patterns for the control tasks (box 2 in Fig. 1). All possible control skip and multirate possibilities are captured as performance and safety-constrained transitions between modes/locations in this finite representation. It generates more fine-grained aperiodic scheduling options compared to the SOTA techniques as it combines CLF-based multirate control switching along with MLF-based control skipping.
3) We devise an algorithmic framework that dynamically observes the current performance degradations of different control loops and accordingly deploys suitable periodic/aperiodic execution patterns to the corresponding control tasks respecting a processor utilization budget (box 4 in Fig. 1).
4) We evaluate our performance-aware dynamic resource-sharing technique in a practical real-world setup and compare it with SOTA approaches to analyze its effectiveness.

To summarize, the theoretically performance-preserving control design guided by resource-aware dynamic scheduling makes this work ideal for efficient bandwidth distribution in resource-limited networked and embedded CPSs.

II. System Model

We express the physical system/plant model as a linear time-invariant (LTI) system having dynamics as follows:

$$\dot{x}(t) = \Phi x(t) + \Gamma u(t) + w(t), \;\; y(t) = Cx(t) + v(t). \qquad (1)$$

Here, the vectors $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and $u \in \mathbb{R}^p$ define the plant state, output, and control input, respectively. $x(t), y(t), u(t)$ are their values at time t. $w(t) \sim \mathcal{N}(0, Q_w)$, $v(t) \sim \mathcal{N}(0, R_v)$ are process and measurement noises. They follow Gaussian white noise distributions with variances $Q_w \in \mathbb{R}^{n\times n}$ and $R_v \in \mathbb{R}^{m\times m}$,

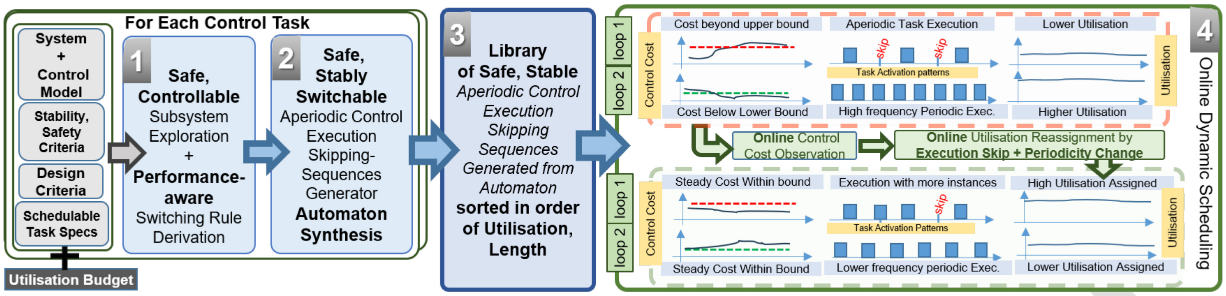

Fig. 1. Overview of the proposed framework.

respectively. The matrices Φ, Γ are the continuous-time state and input-to-state transition matrices, respectively. C is the output transition matrix. Considering that the plant outputs are sampled and control inputs are actuated once in every h sampling interval, we have the following:

$$\hat{x}[k+1] = A\hat{x}[k] + Bu[k] + L\big(y[k] - C\hat{x}[k]\big)$$
$$u[k] = -K\hat{x}[k],\ A = e^{\Phi h},\ B = \int_0^h e^{\Phi t}\Gamma dt. \tag{2}$$

Here, A, B are the discrete-time counterparts of Φ, Γ, respectively. L is the Kalman gain in the Luenberger observer that is used to filter the Gaussian noises from the output and estimate states. K is the feedback control gain designed as per the performance requirements. $x[k], \hat{x}[k], y[k], u[k]$ denote state, estimated state, output and control input vectors at kth ($k \in \mathbb{N}$) sampling instance or $t = kh$ time unit, respectively.

1) Control Design, Performance Metrics, and Safety Criteria: A control design metric represents the control objective while designing the controller. One such standard design metric is *settling time*, i.e., the time that the system takes to maintain a steady output within a fixed error margin around the desired reference value (e.g., within 2% error band). Hence, the controller has to be designed in such a way that the given settling time requirement is always met. We correlate the settling time requirement with the notion of *exponential stability* criteria by defining it as follows.

Definition 1 (Globally Uniformly Exponentially Stable): The equilibrium $x = 0$ of the system in (2) is globally uniformly exponentially stable (*GUES*) if for any initial state $x[k_0]$ there exist $M > 0$, $\gamma < 0$ such that, $\|x[k]\| \leq Me^{\gamma(k-k_0)}\|x[k_0]\|\ \ \forall k \geq k_0$ ($\|.\|$ is vector norm).

Given a settling time requirement of $(k - k_0)h$ (h is the sampling period), this definition mandates the system states *must decay by a minimum factor of e^{γ} at every sampling period* so that the system output stays within a 2% error bound of the desired reference value within $(k - k_0)h$ time. On the other hand, the *control performance* is the measure of the quality of control, i.e., how efficiently the design requirement is met. We consider a linear quadratic regulator (LQR)-based cost function as the performance metric as given below

$$J = \sum_{k=0}^{N-1} (x[k] - r[k])^{\mathsf{T}} Q (x[k] - r[k]) + u^{\mathsf{T}}[k] R u[k])$$
$$+ (x[N] - r[N])^{\mathsf{T}} S (x[N] - r[N]). \tag{3}$$

Equation (3) represents the control cost computed for a system over a finite time-window. Here, the symmetric weighing matrices $Q, R \succ 0$ capture the relative importance that the control designer can give to the state deviation and control effort, respectively. S is the final state cost matrix. Replacing $u[k]$ with $-Kx[k]$, we derive the optimal feedback control gain K that minimizes the control cost J. We tune the state cost matrix Q to design an LQR gain K, that promises to achieve the desired exponential decay γ during the closed-loop evolution, offering the minimum control cost.

The phase difference between the input and output can cause *unsafe* transient behavior (e.g., overshoots, undershoots) even in stable control loops bounded by a GUES criterion. To overcome these, we consider that the bounded region for GUES (see Definition 1) during the control design as an input by the system designer such that a *forward invariance* is maintained. This demands the system states to initialize and always remain within this region. This region is defined as the *safe operating region* $\mathcal{R}_{\text{safe}}$. Formally, the closed-loop system states should always satisfy the following safety criteria: $x(t) \in \mathcal{R}_{\text{safe}} \Rightarrow \forall t' \geq t,\ x(t') \in \mathcal{R}_{\text{safe}}$.

2) Closed-Loop Under Sampling Period Change: We intend to capture both the plant and controller states at each discrete time step by defining an augmented state vector $X = [x^T, \hat{x}^T]^T$. Replacing u with $-K\hat{x}$ and y with Cx [see (2)], the evolution of the overall closed-loop is expressed as follows:

$$X[k+1] = A_{1,h}X[k],\ A_{1,h} = \begin{bmatrix} A & BK \\ LC & A - LC + BK \end{bmatrix}. \tag{4}$$

For multiple sampling periods, h_i, h_j say, we denote the discrete-time augmented system matrices as A_{1,h_i}, A_{1,h_j}. Therefore, such a system that changes its sampling rates can be represented as a switched LTI system, where it switches between different combinations of $\{A_{1,h_i}, A_{1,h_j}\}$.

3) Closed-Loop Under Skipped Control Executions: Because of the uncertain behavior of underlying computing/communication platforms (such as micro-architectural faults, jitters, communication delays, etc.) in embedded control systems, control execution skips may occur while periodically executing control tasks. This causes the ideal periodic control executions to become *aperiodic*, where the actuator on the plant side does not receive any new control update within the sampling interval $[k, k+1)$ or the time interval $[kh, kh+1)$ if a control skips occur at kth time step. Therefore, the value of the control input remains the same as it was in the last sampling instance (last received control actuation at kth instance), i.e., $u[k+1] = u[k]$. Leveraging the robustness available in control loop design accounting for potential actuation misses, we

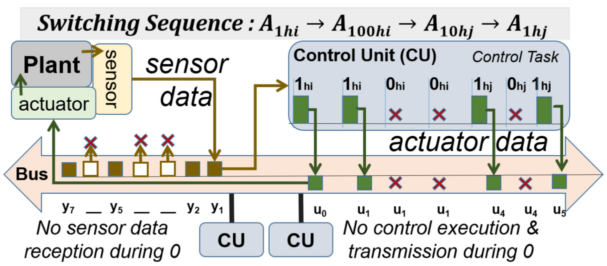

Fig. 2. Closed-loop with skipped control.

consider that some such control task executions and resulting actuation can be intentionally skipped for possible benefit in terms of co-schedulability of control loops. Fig. 2 presents the real-time operation of such a control loop under *intentional control execution skips*. In the presence of a control execution skip at $(k+1)$th instance, the closed-loop system in (2) evolves as $x[k+1] = Ax[k] + Bu[k]$, $\hat{x}[k+1] = I\hat{x}[k] + Ou[k] + Oy[k]$, $u[k+1] = K\hat{x}[k+1]$. I and O are identity and zero matrices with the same dimensions as A, B, respectively. Therefore, the augmented closed-loop system under control execution skip progresses like below

$$X[k+1] = A_{0,h}\, X[k],\;\; A_{0,h} = \begin{bmatrix} A & BK \\ O & I \end{bmatrix}. \tag{5}$$

To assess the system performance under control execution skips, we need to model the system as a discrete-time system (sampling period h) switching between the closed-loop augmented characteristic matrices $A_{1,h}$ from (4) and $A_{0,h}$ from (5).

An example is presented in Fig. 2. Here, after executing the control task at kth and $(k+1)$th instance, we skip the control executions at $(k+2)$th and $(k+3)$th sampling instances, then the augmented closed-loop system state at $(k+4)$th instance becomes $X[k+4] = A_{0,h}A_{0,h}A_{1,h}A_{1,h}X[k]$. Here, we use the control input computed at $t = (k+1)h$ for the next 3 sampling instances, i.e., between the time $t \in [kh, (k+3)h)$. Again, the control input is computed at $t = (k+3)h$ and used up to the time $t \in [(k+3)h, (k+4)h)$ (i.e., the next sampling interval). We express this switching control sequence as aperiodic control execution skipping sequence (ACESS).

Definition 2 (ACESS): An $l \in \mathbb{N}$ length ACESS for a given control loop is a sequence $\rho \in \{0_{h1}, 1_{h1}, \ldots, 0_{hM}, 1_{hM}\}^l$. Here, $\{h1, h2, \ldots, hM\}$ are the sampling periods of its available set of feedback controllers; $\forall h \in \{h1, h2, \ldots, hM\}$, 1_h denotes a control execution when the augmented closed-loop system state X progresses for h time duration following (4), and 0_h denotes a skipped execution when X progresses h duration following (5).

Now, the aforementioned aperiodic control sequence can also be expressed as $1_h 1_h 0_h 0_h$, such that $X[k+4] = A_{0,h}A_{0,h}A_{1,h}A_{1,h}X[k] = A_{0,h}^2 A_{1,h}A_{1,h}X[k]$. Here, the augmented state is periodically controlled at $t = kh$ and $(k+1)h$ and evolves in continuous time following $(A_{0,h})^2 \times A_{1,h}$ until the system states are sampled for the next control execution.

III. Methodology

Our methodology employs *periodic* activation with suitable *control gains* (designed for different sampling rates) along with *aperiodic* activations to the control tasks depending on their performance degradations.

A. Overview

A brief overview of the proposed methodology is illustrated in Fig. 1. The 1,2,3 marked boxes denote the offline part of the methodology, and box 4 denotes the online dynamic scheduling algorithm. The red dotted (top) part inside box 4 shows how two control tasks corresponding to two different control loops are scheduled with certain activation patterns in a shared platform. The green dotted (bottom) part inside box 4 shows the updated activation patterns assigned to these control tasks based on their control cost deviations. For this 2-task setup, observe that control task 1 is scheduled with an ACESS with two skips and control task 2 is scheduled with high-frequency (i.e., small periodicity) periodic activations. Our online algorithm updates their activation patterns following the steps below. 1) It observes whether there is any deviation w.r.t the *performance metric*, e.g., LQR control cost [see (3)] of closed-loop 1 due to some external disturbance. If the deviation goes beyond a tolerable cost margin (marked with a red dotted line in the left side cost plot), an aperiodic activation with fewer skipped executions is assigned (utilizes more bandwidth than before). 2) If the processor utilization goes beyond a fixed budget due to this higher-bandwidth assignment, the algorithm deploys an activation pattern with increased periodicity/lower frequency to control task 2, which consumes a lower bandwidth than before. Doing these requires the following two important components. 1) A library of possible ACESS-s for every control loop so that each of them is safe and *performance-preserving*. This should preferably be given with a finitary representation for resource-constrained implementation (box 3 in Fig. 1). 2) For this, we need a set of switching constraints that ensure stable switching across multirate controllers and ACESS-s (boxes 1 and 2 in Fig. 1). As shown in Fig. 1, from such a library, represented in the form of an automaton for each control loop, our methodology chooses suitable ACESS as well as sampling periods while satisfying the timing constraints.

B. Aperiodic Control Executions—Subsystem View

To generate a sequence of performance-preserving *aperiodic* control activation for a periodic control task, we visualize the control loop as a system, switching between multiple subsystems. Each subsystem represents the control loop under a certain number of consecutively skipped control executions. We consider this evolution of an augmented closed-loop discretized with a fixed sampling period, under aperiodic control execution, as a control skipping subsequence (CSS). The subsystems are chosen such that their time and frequency domain characteristics always ensure safe transient behavior (i.e., by confining system states within $\mathcal{R}_{\text{safe}}$).

Definition 3 (CSS): A CSS in an l-length ACESS ρ is an i-length subsequence having the form $1_h(0_h)^{i-1}$, $i \le l$. Like ACESS, 1_h signifies the execution of a controller designed with the sampling period h and its augmented state progression following (4). The following 0_hs signify the actuation of the last control input once in each of the next $(i-1)$ sampling instances (i.e., $(i-1)h$ time) such that the states evolve at every sampling instance following (5).

Example 1: Consider an $l = 7$ length ACESS $\rho = 1_{hi}1_{hi}0_{hi}0_{hi}1_{hj}0_{hj}1_{hj}$ that spans for 7 sampling instances, i.e., during $t \in [khi, (k+4)hi + 3hj)$ it follows the CSSs, 1_{hi}, at position $\rho[0]$ signifying a control execution and state evolution using $A_{1,hi}$, $1_{hi}0_{hi}0_{hi}$ at positions $\rho[1], \rho[2], \rho[3]$ signifying a control execution and state evolution using $A_{100,hi} = A^2_{0,hi}A_{1,hi}$, $1_{hj}0_{hj}$ at $\rho[4], \rho[5]$, signifying a control execution and state evolution using $A_{10,hj} = A^1_{0,hj}A_{1,hj}$ and 1_{hj} at $\rho[6]$ signifying a control execution and state evolution using $A_{1,hj}$ (see Fig. 2). Note, corresponding to each such i-length CSS, $1_h0_h^{i-1}$, the controller executes once followed by the continuous evolution of the plant over a time window of $i \times h$ instead of h, where the same control input is actuated once every h. Hence, $X[7] = A_{1,hj}A_{10,hj}X[4] = A_{1,hj}A_{10,hj}A_{100,hi}A_{1,hi}X[0] = (A_{0,hj})^0 A_{1,hj}(A_{0,hj})^1 A_{1,hj}(A_{0,hi})^2 A_{1,hi}(A_{0,hi})^0 A_{1,hi}X[0]$.

To generalize, for an l-length ACESS, $\rho = 1_{h1}0_{h1}^{i_1-1}1_{h1}0_{h1}^{i_2-1}1_{h2}0_{h2}^{i_3-1}\cdots 1_{hM}0_{hM}^{i_N-1}$, such that $l = \sum_{q=1}^{N} i_q$ (i.e., ρ contains total N CSSs), a closed-loop dynamical system, discretized with a set of sampling periods $\{h1, h2, \ldots, hM\}$, behaves like a switched system having state evolution of the form: $x[l] = (A_{0,hM})^{i_N-1}A_{1,hM}\ldots(A_{0,h1})^{i_2-1}A_{1,h1}(A_{0,h1})^{i_1-1}A_{1,h1}x[0]$. Note that each q-length CSS $1_h0_h^{q-1}$ in an ACESS (for any $q \in \mathbb{Z}^+$) is defined for a certain sampling period h. This essentially represents a closed loop that actuates the control input once in every h duration for qh duration and the control input is computed by a controller designed with h sampling period. We perceive this as a *subsystem* $\theta_{q,h}$, for any $q \in \mathbb{Z}^+$. For example, under the ACESS $1_{hi}1_{hi}0_{hi}0_{hi}1_{hj}0_{hj}1_{hj}$ as demonstrated in Example 1, the closed-loop evolves according to the following subsystem switching sequence: $\theta_{1,h_i} \to \theta_{3,h_i} \to \theta_{2,h_j} \to \theta_{1,h_j}$.

C. Stability Analysis of ACESS-s as Switched Systems

Let $\Theta = \{\theta_1, \theta_2, \ldots\}$ be a set of subsystems where any qth subsystem $\theta_q = \theta_{i,h}$ represents the dynamics of a closed-loop system with a sampling period $h \in \mathcal{H}$ and following a CSS 10^{i-1}, $i \in N$. Here, $\mathcal{H}, N$ are a set of chosen periodicities and lengths of CSSs, respectively. To enable switching between all possible combinations of sampling periods and CSSs, we define $q = \langle i, h\rangle, i \in N, h \in \mathcal{H}$, such that all possible combinations from $N \times \mathcal{H}$ are present in Θ. Each of these subsystems is linear in nature [follows (4) and (5)]. We present the detailed stability analysis for a switched system comprising subsystems in Θ in a discrete-time setting. We represent such a system as a switched linear system like the one below

$$X[k+1] = A_{\sigma[k]}(X[k]),\ \sigma : k \mapsto \mathbb{N},\ k \geq 0. \tag{6}$$

Here, σ is a switching signal. $\sigma(k) = q$ signifies that at kth sampling instance the system is in qth subsystem θ_q, i.e., X evolves using a controller designed with the sampling period h and following a CSS 10^{i-1}, when $q = \langle i, h\rangle$. We define $N_{\sigma_q}(k', k)$ as the number of switching to θ_q within k'th and kth sampling instances. Thus

$$N_{\sigma_q}(k', k) \leq N_{0q} + T_q(k', k)/\tau_{d_q} \tag{7}$$

where N_{0q} is chattering bound for θ_q, $T_q(k', k)$ is the total time spent in θ_q and τ_{d_q} is a minimum time duration that the switched system stays in the subsystem θ_q. Note that the minimum time that the switched system dwells in every subsystem is subsystem-specific. This minimum dwelling duration is known as mode-dependent average dwell time (MDADT) [15]. The MDADT for θ_q is denoted with τ_{d_q}. For constraining the MDADTs of a set of subsystems (e.g., Θ) to ensure a desired performance bound while switching within them, we use MLFs.

Let there exist a radially unbounded, continuously differentiable, positive definite function, $V_q(x(k))$, such that $V_q : \mathbb{R}^n \mapsto \mathbb{R}$ for all $\theta_q \in \Theta$. If there exist class $\mathcal{K}_\infty$ functions κ_1, κ_2, and switching values $\sigma(k_p) = q$, $\sigma(k_p^-) = q'$, where $k_p^- < k_p$, $q \neq q'$, $\theta_q, \theta_{q'} \in \Theta$ are two different subsystems, then $\forall \theta_q \in \Theta$ the following holds:

$$\kappa_1(||x(k)||) \leq V_q(x(k)) \leq \kappa_2(||x(k)||) \tag{8}$$

$$\Delta V_q(x(k)) \leq \alpha_q\ V_q(x(k)) \quad \text{s.t.} \alpha_q \neq 0 \tag{9}$$

$$V_q(x(k)) \leq \mu_q\ V_{q'}\big(x(k^-)\big)\ \forall \theta_{q'} \in \Theta \text{ for } \mu_q > 1. \tag{10}$$

In simpler words, V_q-s are subsystem wise MLFs respecting (8)–(10). In (9), for stable subsystems, $0 < 1 + \alpha_q < 1$ and for unstable subsystems $1 + \alpha_q > 0$, where α_q is a function of the minimum attainable exponential decay by θ_q [15]. Extending the MDADT-based asymptotic stability criteria from [15, Th. 2], we claim the following in order to maintain the desired GUES while slowly switching between subsystems in Θ.

Claim 1: For the switched system in (6) having subsystems with MLF $V_q(X[k])$ satisfying (8)–(10), the following criteria need to be satisfied in order to ensure a desired GUES margin γ while slowly switching between a set of subsystems: 1) for every qth subsystem, there exists a lower bound of the corresponding *MDADT* $\tau_{d_q} \geq (\ln \mu_q/[|\ln(1+\alpha_q)|])$ and 2) the switching should follow a minimum dwell time ratio $\nu = ([\ln\gamma^+ - \ln\gamma]/[\ln\gamma - \ln\gamma^-])$ between the total dwelling duration at stable and unstable subsystems, where $\gamma^- = \max_{q\in\Theta^-}[(1+\alpha_q)\mu_q^{(1/\tau_{d_q})}]$ and $\gamma^+ = \max_{q\in\Theta^+}[(1+\alpha_q)\mu_q^{(1/\tau_{d_q})}]$.

Proof: From (9) and (10), for $k \in [k_p, k_{p+1})$ we can write

$$V_{\sigma(k)}(x(k)) \leq \big(1 + \alpha_{\sigma(k_p)}\big)^{T_{\sigma(k_p)}(k_p,k)} \mu_{\sigma(k_p)} V_{\sigma(k_p^-)}\Big(x(k_p^-)\Big). \tag{11}$$

Unwinding (11) over the switching interval $[k_0, k)$ will have the parameters μ_q and α_q repeated in the above equation as many times as the qth subsystem θ_q will be switched into. Hence, using the definition of *MDADT* in (7), and the total number of subsystems in Θ as $\mathcal{M}$, we can rewrite the above equation as we get

$$\begin{aligned} V_{\sigma(k)}(x(k)) \leq\ & \mu_{\sigma(k)}^{N_{\sigma(p)}(k_p,k)}\big(1+\alpha_{\sigma(p)}\big)^{T_{\sigma(p)}(k_p,k)}\mu_{\sigma(k_p)}^{N_{\sigma p}(k_{p-1},k_p)} \\ & \big(1+\alpha_{\sigma(p-1)}\big)^{T_{\sigma(p-1)}(k_{p-1},k_p)}\ldots\mu_{\sigma(1)}^{N_{\sigma(0)}(k_0,k_1)} \\ & \big(1+\alpha_{\sigma(0)}\big)^{T_{\sigma(0)}(k_0,k_1)}V_{\sigma(k_0)}(x(k_0))\mu_{\sigma(k)}^{N_{\sigma_q}(k_0,k)} \end{aligned}$$

$$\leq \prod_{q=1}^{\mathcal{M}} \left(\mu_q^{N_{\sigma_q}(k_0,k)} (1+\alpha_q)^{T_q(k_0,k)} \right) V_{\sigma(k_0)}(x(k_0))$$

$$\leq e^{\left\{ \sum_{q=1}^{\mathcal{M}} N_{0q} \ln \mu_q \right\}} \prod_{q\in\Theta^-} \left((1+\alpha_i)\mu_q^{\frac{1}{\tau_{d_q}}} \right)^{T_q(k_0,k)}$$

$$\prod_{q\in\Theta^+} \left((1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}} \right)^{T_q(k_0,k)} V_{\sigma(k_0)}(x(k_0)).$$

If we set, $\mathcal{K} = e^{\{\sum_{i=1}^{\mathcal{M}} N_{0q} \ln \mu_i\}}$ and

$$T^- = \sum_{q\in\Theta^-} T_i(k_0,k), \gamma^- = \max_{q\in\Theta^-} \left[\ln\left[(1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}} \right] \right] \quad (12)$$

$$T^+ = \sum_{q\in\Theta^+} T_q(k_0,k), \gamma^+ = \max_{q\in\Theta^+} \left[\ln\left[(1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}} \right] \right] \quad (13)$$

$$\text{Hence, } \left(e^{\gamma^- T^-} \times e^{\gamma^+ T^+} \right) \leq e^{\gamma(k-k_0)}, \quad (14)$$

$$\Rightarrow V_{\sigma(t)}(x(t)) \leq \mathcal{K} e^{(-\gamma^- T^- + \gamma^+ T^+)} V_{\sigma(k_0)}(x(k_0))$$

$$\leq \mathcal{K} e^{[\gamma(k-k_0)]} V_{\sigma(k_0)}(x(k_0)). \quad (15)$$

Since $\gamma^- \leq \gamma < 0 \quad \forall q \in \Theta^- \; 1 > \gamma^- > \gamma^{-^2} \geq (1+\alpha_q) > 0$

$$\mu_q^{\frac{1}{\tau_{d_q}}} \leq \frac{1}{(1+\alpha_q)} \quad \Rightarrow \tau_{d_q} \geq \frac{\ln \mu_q}{|\ln(1+\alpha_q)|}. \quad (16)$$

Since $\gamma^+ > 1$ and $\forall q \in \Theta^+$, $0 < \alpha_q \Rightarrow 1 < (1+\alpha_q)$

$$\mu_q^{\frac{1}{\tau_{d_q}}} \geq \frac{1}{(1+\alpha_q)} \quad \Rightarrow \tau_{d_q} \geq -\frac{\ln \mu_q}{\ln(1+\alpha_q)}. \quad (17)$$

Note that T^- and T^+ represent the total running time into the stable and unstable subsystems, respectively. Therefore, from (14), we have the *dwell time ratio*, ν, between the stable and unstable subsystems as, $\nu = (T^-/T^+) \geq ([\gamma^+ - \gamma]/[\gamma - \gamma^-])$. From (15) and the definition of GUES [6], we can conclude that $V_{\sigma(k)}(x(k))$ converges to zero with the desired margin of γ as sampling instance $k \to \infty$, and consequently we get the lower bound of the *MDADT* τ_{d_q} for $q \in \{1, 2, \ldots, \mathcal{M}\}$. ∎

For arbitrary switching, the following must hold.

Claim 2: A switched system in (6) that arbitrarily switches between subsystems should have a CLF for all its switchable subsystems that satisfy (8)–(9) and $V_q(x(k)) = V_{q'}(x(k^-)) \quad \forall \theta_q, \theta'_q \in \Theta_{clf}$ s.t. $\Theta_{clf} \subseteq \Theta$ in order to maintain the desired GUES decay margin of γ while switching among them arbitrarily [12].

Proof: The proof of this theorem can be established following the previous proof, considering $\mu_q = 1$ for all qth subsystem in Θ_{clf}, which is a subset of all subsystems Θ (mandates a *zero* dwell time or arbitrary switching to and from each subsystem in Θ_{clf}) [12]. ∎

D. Computing Stable Switching Rules for Safe Subsystems

Once we have chosen sets of periodicities for each control task such that their corresponding discrete-time closed loop systems are controllable, we can design stabilizable LQR controllers and Kalman gains. In each case, it must be ensured

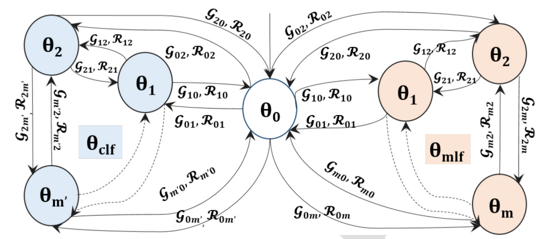

Fig. 3. Schematic of CSA.

that the augmented states of these closed loops (i.e., actual and estimated states) remain within a safe operating region $\mathcal{R}_{\text{safe}}$. For each of the sampling periods for a system, we can find the maximum bound of CSS length for which the safety property is maintained. To achieve the desired decay rate of γ while switching between multiple subsystems in Θ, we can calculate required *MDADT* τ_{d_q}, using (16), $\forall q \in N \times \mathcal{H}$ such that $X[k] \in \mathcal{R}_{\text{safe}} \quad \forall k, k_0 \in \mathbb{Z}^+, k \geq k_0$, i.e., the augmented state always remains confined within a common *safe operating region* for a given control loop. We consider *quadratic* MLF candidates for all subsystems, i.e., $V_q = X^T P_q X$, where $P_q > 0$. To achieve this, in accordance with Claim 1, we need to estimate a minimum possible $\mu_q > 1$ such that the following linear matrix inequalities (LMIs) have a positive definite solution for P_q, given the values of $\alpha_q = \lambda^2_{\max,q} - 1, \quad \forall \theta_q, \theta_{q'} \in \Theta$. Here, $\lambda_{\max,q}$ is the maximum Eigenvalue of discrete-time subsystem θ_q

$$A_q^T P_q A_q - P_q \leq \alpha_q P_q, \; P_q \leq \mu_q P_{q'}, \; P_q > 0. \quad (18)$$

Therefore, if we switch between the subsystems respecting the derived MDADT and the dwell time ratio (when there are systems with a lower-decay rate than desired), it is guaranteed to maintain the desired GUES. The sets of subsystems for which there exists a valid $P > 0$ for solving the LMIs in (18) with each $\mu_q = 1$ (i.e., share a CLF) can switch between themselves arbitrarily. Since each of the subsystems is chosen such that they keep the system within a safe operating region, the switching is also expected to maintain the desired *safety*.

E. Formalizing Stable Switching Rules as Automaton for Safe and Stable ACESS Generation

Let, $\Theta = \Theta_{clf} \cup \Theta_{mlf}$, where Θ_{clf} is the set of subsystems that have a CLF (supports Claim 2) and Θ_{mlf} is the set of subsystems that can only have MLFs (does not have a CLF and supports Claim 1) given the GUES decay rate γ. Note that some subsystem θ_q can belong to both in Θ_{clf} and Θ_{mlf} since they may support fast switching within one set of subsystems ($\in \Theta_{clf}$) and slow switching between different sets of subsystems ($\in \Theta_{mlf}$) respecting the stability and safety criteria (see Fig. 3). Solving the constraint satisfaction problem explained in Section III-D (see (18)), we calculate the *MDADT* τ_{d_q} for each such subsystem $\theta_q \in \Theta_{mlf}$ (that we denote with $\theta_{q,mlf}$) and the *dwell time ratio* ν. For $\theta_q \in \Theta_{clf}$ (that we denote with $\theta_{q,clf}$), the required minimum dwell time is the same as the time span of the corresponding CSS, i.e., ih for 10^{i-1} if $q = \langle i, h \rangle$, where h is the sampling period for θ_q.

All possibilities of rule-based switching between subsystems can be captured in the form of a finite state generator automaton. The traces of this automation are essentially timed sequences of switching among subsystems or CSS switching sequences that maintain the required decay by ensuring *dwell time* and *dwell time ratio*, respectively. Since such a CSS/subsystem switching sequence generates a safe and stable control execution skipping sequence, we term it CSA. In Fig. 3, we provide a schematic structure for this CSA $\mathcal{T}$ realized with m' stabilizable locations in Θ_{clf} and m locations in Θ_{mlf}. By reusing the notations of the subsystems as the notations for the locations to represent their equivalence, we define the CSA as follows.

Definition 4 (CSA): A CSA for a control loop is a finite state automaton $\mathcal{T} = \langle \mathcal{L}, \{\theta_0\}, \{\theta_0\}, \mathcal{C}, V, \mathcal{E}, Inv\rangle$ where as follows.

1) $\mathcal{L} = \Theta \cup \{\theta_0\}$ is the finite set of locations that denotes the underlying subsystems.
2) θ_0 is the only member of the set of initial locations and the set of accepting locations. It is a dummy subsystem that contributes to the ACESS-s with 0 length and 0 minimum dwell time and helps to synthesize ACESS-s starting from (ending at) any subsystem.
3) $\mathcal{C} = \{c, c', p, p'\} \in \mathbb{R}^+$ are the set of real-valued variables that are used to keep track of real-time during system progression, termed as *clocks*. The clocks c, c', p, p' are used to keep track of global time, local time at each location/subsystem, total dwelling time in stable subsystems, and in unstable subsystems, respectively.
4) $V = \{a, b, s\} \in \mathbb{Z}^+$ is a set of variables other than clocks that are used to keep track of total ACESS length, the count of 0s, and whether the destination subsystem is stable ($s = 0$) or not ($s = 1$), respectively.
5) $\text{Inv}(\theta_q) = \langle \text{Inv}^q_{\text{safety}}, \text{Inv}^q_{\text{len}}, \text{Inv}^q_{dtr}\rangle$ is the invariant tuple at the qth location/subsystem. For $\text{Inv}^q_{\text{safety}}$ and $\text{Inv}^q_{\text{len}}$, the subscripts refer to the rule it enforces at each $\theta_q \in \mathcal{L}$. Inv_{dtr} enforces a maximum dwell time ratio if θ_q is unstable such that a desired *ACESS* length is respected. All of them should hold *true* to stay in θ_q.
6) $\mathcal{E} \subseteq \{(\mathcal{L}\setminus\Theta_{mlf}) \times \mathcal{G} \times \mathcal{R} \times (\mathcal{L}\setminus\Theta_{mlf})\} \cup \{(\mathcal{L}\setminus\Theta_{clf}) \times \mathcal{G} \times \mathcal{R} \times (\mathcal{L}\setminus\Theta_{clf})\}$ is the set of transitions/edges. A transition from θ_q to θ'_q is denoted by, $\mathcal{E}_{qq'} = (\theta_q, \mathcal{G}_{qq'}, \mathcal{R}_{qq'}, \theta_{q'}) \in \mathcal{E}$, where $\mathcal{G}_{qq'}$ represents guard condition and $\mathcal{R}_{qq'}$ is the reset map. Note that there exists no $\mathcal{E}_{00} \in \mathcal{E}$.
7) The guard conditions are defined as

$$\mathcal{G}_{qq'} = \begin{cases} \mathcal{G}^{qq'}_{\text{len}} \wedge \mathcal{G}^{qq'}_{\tau_d} \wedge \mathcal{G}^{qq'}_{\text{safe}} \wedge \mathcal{G}^{qq'}_{\Theta+} & \text{when } \theta_q, \theta'_q \in \mathcal{L} \setminus \Theta_{clf} \\ \mathcal{G}^{qq'}_{\text{len}} \wedge \mathcal{G}^{qq'}_{\tau_d} \wedge \mathcal{G}^{qq'}_{\text{safe}} & \text{when } \theta_q, \theta'_q \in \mathcal{L} \setminus \Theta_{mlf}. \end{cases}$$

Here, $\mathcal{G}^{qq'}_{\text{len}}$ is true when the transition between two locations in $\Theta_{mlf} \cup \theta_0$ or between two locations in $\Theta_{clf} \cup \theta_0$ is possible, respecting the desired length and other requirements, i.e., MDADT of the destination and *dwell time ratio*. If the MDADT of θ'_q cannot be covered or the *dwell time ratio* cannot be maintained (in case of unstable θ'_q), for the desired length of ACCESS, $\mathcal{G}^{qq'}_{\text{len}}$ evaluates to *false*. A *true* value of $\mathcal{G}^{qq'}_{\tau_d}$ denotes that the minimum dwell time required for $\theta_{q,clf}$ or $\theta_{q,mlf}$ is maintained during the transition from θ_q to θ'_q. Note that the minimum dwell time for a $\theta_{q,clf}$ is ih if $\theta_q = \theta_{i,h}$ (for θ_0 it is 0). $\mathcal{G}^{qq'}_{\Theta+}$ is evaluated when the destination location θ'_q is unstable. It evaluates to *true* if there is enough length remaining to maintain the *dwell time ratio* when there is a transition to an unstable subsystem. The variables from V are used to evaluate the length, the local clock c' is used to maintain the *MDADT* related guard, and *dwell time ratio* related guards use $p, p' \in \mathcal{C}$.
8) The reset maps are defined as, $\mathcal{R}_{qq'} = \mathcal{R}^{qq'}_{\text{len}} \wedge \mathcal{R}^{qq'}_{dtr} \wedge \mathcal{R}^{qq'}_{s} \wedge \mathcal{R}^{qq'}_{\tau_d}$ Here, $\mathcal{R}^{qq'}_{\text{len}}$ is used to update the $a, b \in V$ to count the length and number of zeros in the ACESS during the current transition. $\mathcal{R}^{qq'}_{s}$ resets the indicator variable $s \in V$ if $\theta_{q'}$ is stable or θ_0 and sets it otherwise. Finally, $\mathcal{R}^{qq'}_{dtr}$, $\mathcal{R}^{qq'}_{\tau_d}$ update $p, p' \in \mathcal{C}$ and $c, c' \in \mathcal{C}$ to maintain current dwell time ratio and dwelling time of θ_q during the current transition, respectively.

CSA fundamentally is an extended version of *timed automata* that uses discrete variables along with real-valued clocks [17]. From a language theoretic point of view, we designate θ_0 as the starting and accepting state of the automaton by ensuring the guard $\mathcal{G}_{0q}, \mathcal{G}_{q0} \quad \forall \theta_q \in \Theta \setminus \theta_0$. These location invariants and transition guards are generated based on the computations done for a closed loop control task in Section III-D. Therefore, *an ACESS satisfying the original GUES requirement γ within a safe operating region* $\mathcal{R}_{\text{safe}}$ is essentially created by concatenating the CSSs corresponding to each subsystem transitioned through in any cyclic trace in this *control skipping extended timed automaton* CSA starting and ending at θ_0.

Example 2: Consider a CSA with set of locations $\{\theta_0, \theta_{1,mlf}, \theta_{2,mlf}, \theta_{3,mlf}, \theta_{4,mlf}, \theta_{1,clf}, \theta_{2,clf}, \theta_{3,clf}\} \in \mathcal{L}$, where $\theta_1 = \theta_{1,0.02}$, $\theta_2 = \theta_{2,0.02}$, $\theta_3 = \theta_{2,0.04}$ and $\theta_4 = \theta_{3,0.02}$. Among these, the first three have a CLF for a GUES decay rate. The MDADTs for $\theta_1, \theta_2, \theta_3, \theta_4$ are 0.06, 0.08, 0.08, 0.06 sec, respectively, while the *dwell time ratio* is 1.5 to maintain the given GUES.

CSA Trace 1: A sample CSA trace of length 7, generated by slowly switching among the subsystems/locations $\in \Theta_{mlf}$ along with its corresponding ACESS and task activations, are shown in Fig. 4. The topmost plot in Fig. 4 denotes this control task activation sequence of length 7 that is generated from this slow switching. Note that the task instances arrive in every 0.02s interval, denoting a control execution with sampling periodicity $h = 0.02$ s. At the last two arrival instances, the executions are skipped, denoting the corresponding ACESS to be "$(1_{0.02})^4 1_{0.02} (0_{0.02})^2$". As shown in the middle plot of Fig. 4 the system follows the CSS $1_{h-0.02}$ of length $i = 1$ for

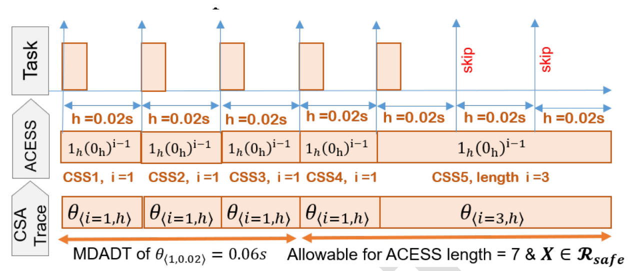

Fig. 4. ACESS with MDADT constraint: Trace 1 in Example 2.

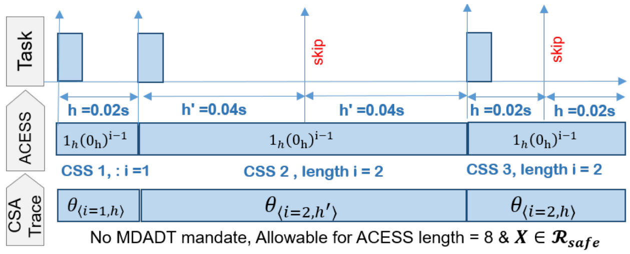

Fig. 5. ACESS with multirate + CLF constraint: Trace 2 in Example 2.

4 sampling iterations and follows the CSS $1_{h-0.02}(0_{h-0.02})^2$ of length $i = 3$ for the next 3 sampling iterations. This corresponds to the cyclic CSA trace $(\theta_0, a, b, c, c', p, p', s = 0) \rightarrow (\theta_{1,mlf}, c = 0.1, c' = 0.1, a = 4, b = 0, p = 0.08, p' = 0, s = 1) \rightarrow (\theta_{4,mlf}, c = 0.16, c' = 0.06, a = 7, b = 2, p = 0.14, p' = 0, s = 1) \rightarrow (\theta_0, c = 0.16, c' = 0, a = 7, b = 2, p = 0.1, p' = 0.06, s = 0)$. The bottom-most part of Fig. 4 shows this trace without the starting and ending θ_0. Starting from θ_0, the system spends $c' = 0.08$ s in θ_1, which evaluates to $(1_{0.2})^4$. Following this, it enters subsystem Θ_4, since 1) as per $\mathcal{G}^{14}_{\text{len}}$ this does not violate the desired length, i.e., value$(a) + 3 \leq 7$ (remember $\theta_4 = \theta_{3,0.02}$ which evaluates to CSS $1_{0.02}(0_{0.02})^2$ of length $i = 3$); 2) as per the dwell time constraint $\mathcal{G}^{14}_{\tau_d}$, value$(c') = 0.08 > \tau_{d_1} = 0.6$; and 3) as per $\mathcal{G}^{14}_{\text{safety}}$, $\text{Inv}^4_{\text{safety}}$, the augmented system states are within a safe operating region in that time window. Then, CSA transits to θ_0 as value$(a) == 7$ (i.e., $\mathcal{G}^{40} = $ true) and we get the ACESS $(1_{0.02})^4 1_{0.02}(0_{0.02})^2$.

CSA Trace 2: Another task execution sequence corresponding to an ACESS of length 8 is shown in Fig. 5. The ACESS is generated from the CSA by fast switching between subsystems $\in \Theta_{clf}$. Here, the closed-loop system switches between a controller with sampling period of 0.02 s and a controller with sampling period of 0.04 s. For both, it deploys $(1_{h=0.04})0_{h=0.04})^1$ and then $(1_{h=0.02})0_{h=0.02})^1$ as CSSs. Corresponding, cyclic timed trace of this CSA is $(\theta_0, a, b, c, c', p, p', s = 0) \rightarrow (\theta_{1,clf}, c, c' = 0.08, a = 4, b = 0, p = 0.08, p' = 0, s = 0) \rightarrow (\theta_{3,clf}, c = 0.2, c' = 0.12, a = 8, b = 2, p = 0.12, p' = 0, s = 0) \rightarrow (\theta_0, c = 0.2, c' = 0, a = 8, b = 0, p = 0.2, p' = 0, s = 0)$.

In the following section, we present an algorithm that generates ACESS-s for each control task scheduled in a shared execution platform from their CSAs.

F. Algorithmic Framework for Task-Wise ACESS Scheduling

A CSA generates all possible safe and stable *ACESS-s* for a control task given with *certain performance criteria and length requirements*. Considering the subsystems/locations as vertices along with their transitioning edges, the CSA can be imagined as an almost complete graph with $|\Theta|$ nodes and $|\mathcal{E}|$ many edges (complete, since all vertices are connected but some edges may be infeasible based on guards). Hence, an algorithm designed to look for a valid trace (i.e., *safe and stable* ACESS) of such a finite state automaton using DFS takes $O(|\Theta| + |\mathcal{E}|) \approx O(n^2_{\max})$ time. Here, $n_{\max}$ is the maximum possible subsystem count for a controller. To achieve a linear order implementation of this algorithm, we do the following offline preprocessing.

Algorithm 1 Performance-Aware Dynamic Task Scheduling

Input: Task set $TS = \{T_1, T_2, \cdots, T_{\mathcal{K}}\}$, $\mathcal{M}^{(j)}$, $\{\Theta^{(j)}_0, \forall T_j \in \text{TS}\}$, task specs. $Specs = \{spec^{(j)} | spec^{(j)} = \langle h^{(j)}, \rho^{(j)}, c^{(j)}, J^{(j)}, J^{(j)}_{ub}, J^{(j)}_{lb} \rangle \forall T_j \in TS\}$, utilization budget U_b
Output: Updated *Specs* with schedulable ACESS-s $\{\langle h^{(j)}, \rho^{(j)} \rangle \forall T_j \in TS\}$
$utils, newUtils \leftarrow [\,], [\,]$
$sort(TS, priority, descending)$
for each $j < \mathcal{K}$ **do**
  $utils[j] \leftarrow$ GETUTIL(T_j) ▷ compute task wise Utils.
  $J^{(j)} \leftarrow$ LQR cost for current states of τ_j following Eq. (3)
  if $J^{(j)} \geq J^{(j)}_{ub}$ **then**
    $newUtils[j] \leftarrow utils[j] \times \frac{J^{(j)}}{J^{(j)}_{ub}}$ ▷ util. increment for high cost
  else $newUtils[j] \leftarrow utils[j]$
  end if
end for
if SUM$(newUtils) \geq U_b$ **then** ▷ If utilisation beyond budget
  $j' \leftarrow \mathcal{K}$ ▷ start from lower priority tasks
  for each $j' > 0$ **do**
    if $J^{(j')} \leq J^{(j')}_{lb}$ **then**
      $newUtils[j'] \leftarrow utils[j'] \times \frac{J^{(j')}}{J^{(j')}_{lb}}$ ▷ reduce util. if low cost
    end if
  end for
end if
if SUM$(newUtils) \geq U_b$ **then** ▷ If util. still beyond budget
  $utilAdjFact \leftarrow U_b / \text{SUM}(newUtils)$
  $newUtils \leftarrow utilAdjFact \times newUtils$ ▷ scale down task wise Util.
end if
for each $T_j \in TS$ **do**
  $\Theta_0[j] \leftarrow$ GETNEXTLOC$(\Theta^{(j)}_0, \rho^{(j)})$ ▷ next switchable subsystems
  if $newUtils[j] > utils[j]$ **then**
    $\mathcal{M}[j] \leftarrow$ GETACESSMIN$(\mathcal{M}^{(j)}, newUtils[j], \Theta_0[j], len(\rho^{(j)}))$
  else $\mathcal{M}[j] \leftarrow$ GETACESSMAX$(\mathcal{M}^{(j)}, newUtils[j], \Theta_0[j], len(\rho^{(j)}))$
  end if
end for
$HP \leftarrow findMinCommonHP(\mathcal{M})$ ▷ find common HP for *TS*
for each $T_j \in TS$ **do**
  $Specs[j].\langle h^{(j)}, \rho^{(j)} \rangle \leftarrow \mathcal{M}[j][HP][0]$ ▷ deployable new ACESS-s
end for
return *Specs*

1) Offline Preprocessing With Practical Assumptions: 1) For a control task $T_j \in TS$, we store the set of arbitrarily switchable subsystems starting from a current subsystem $\theta_{(h^{(j)}, n^{(j)})} \in \Theta_{clf}$ following Claim 2 in $\Theta_{(h^{(j)}, n^{(j)})}$. This collection of single step reachable subsystem set $\Theta_{(h^{(j)}, n^{(j)})}$ $\forall T_j \in TS$, is stored in a hash map Θ_0, where the key is the current subsystem and the value contains the set of subsystems. $\Theta^{(j)}_0 = \{\langle (h^{(j)}, n^{(j)}) : \Theta_{(h^{(j)}, n^{(j)})} \rangle \cdots \forall n^{(j)} \in N^{(j)}, h^{(j)} \in \tilde{\mathcal{H}}^{(j)}\}$. We define a method GETNEXTLOC$(\boldsymbol{\Theta^{(j)}_0}, \rho_j)$ that outputs the set of subsystems from this hash map $\boldsymbol{\Theta^{(j)}_0}$ (input), that can be arbitrarily switched into, starting from the last visited subsystem of an ACESS ρ_j (input) in $O(1)$.

2) There exist different ACESS length choices for each $T_j \in TS$ given a fixed hyperperiod choice, corresponding to the sampling period choices of T_j. We can generate all possible ACESS-s for each of these length choices from the CSA $\mathcal{T}^{(j)}$ given a hyper-period (HP) choice. We compute their processor utilization and group them w.r.t their processor utilization. Each group is then sorted in ascending order of their utilizations. We further group these ACESS-s w.r.t. their starting subsystem (after θ_0). Each group with the same utilization and starting subsystem is again subgrouped w.r.t their HPs and sorted by the ascending order of HP-duration. The resulting data structure $\mathcal{M}^{(j)}$ is a map of maps. The tuple containing

utilization (*util*) and starting subsystem in each ACESS-s (i.e., after θ_0 in CSA) is the *key* and another list of maps are *values* in the outer map. The inner map uses HPs as *key*, and the ACESS length (*len*), the corresponding set of ACESS-s as *values* like following: $\mathcal{M}^{(j)} = \{\langle \text{util} \downarrow_1,\ \theta_{\text{start}} \rangle : \{HP \downarrow_2 : \{\langle len \downarrow_3, \{\rho_1, \rho_2 \cdots\}\rangle, \ldots, \}, \ldots, \}, \ldots, \}$. The outer list of maps $\mathcal{M}^{(j)}$ is sorted in ascending order of *util*, denoted by $\downarrow_1$. The inner list of maps is sorted in the ascending order of *HP*, denoted by $\downarrow_2$. The values against each key in the inner maps are sorted w.r.t. the length of the ACESS, *len*, denoted by $\downarrow_3$.

We create method GETACESSMIN($\mathcal{M}^{(j)}, \text{util}_{\min}, \theta_{\text{start}}, \text{len}_{\min}$) to generate the list of all possible ACESS-s with a *minimum utilization* $\text{util}_{\min}$ (input), starting from the subsystem θ_{start} (input) and having a *minimum length* of $\text{len}_{\min}$ (input), grouped by HPs from $\mathcal{M}^{(j)}$ (input). Another method GETACESSMAX($\mathcal{M}^{(j)}, \text{util}_{\max}, \theta_{\text{start}}, \text{len}_{\max}$) generates the list of all possible ACESS-s with a *maximum utilization* $\text{util}_{\max}$ (input), starting from the subsystem θ_{start} (input) and having a *maximum length* $\text{len}_{\max}$ (input) grouped by HPs from $\mathcal{M}^{(j)}$ (input). Considering that the maximum number of entries of utilization-keys in $\mathcal{M}^{(j)}$ is $m_{\max}$, both these methods take $O(\log(m_{\max}))$ time since the maps $\mathcal{M}^{(j)}$s are sorted w.r.t utilization. Note that we can fetch the ACESS-s corresponding to θ_{start} or a *HP* key in $O(1)$, resulting in an overall $O(\log(m_{\max}))$ complexity. Now, we present an online algorithm to dynamically schedule ACESS-s to control tasks based on their performance degradations using these data structures.

2) Performance-Aware Dynamic Scheduling: For a feedback-driven ACESS deployment from CSA for each of these control tasks, we propose Algorithm 1. Performance degradation caused by external disturbances increases the control cost of a task. In such scenarios, the controller needs an increased number of executions (i.e., higher frequency/more 1's in the corresponding ACESS-s) to reduce the performance degradation and, thereby, the control cost. Consequently, it demands a higher-processor utilization. To achieve this, Algorithm 1 takes the following inputs.

1) The set of $\mathcal{K}$ control tasks $TS = T_1, T_2, \ldots, T_{\mathcal{K}}$ that are to be scheduled dynamically in a given platform.
2) A list of ACESS-s $\mathcal{M}^{(j)}$ for each $T_j \in TS$ sorted in the order of utilization, length, etc.
3) The list of arbitrarily switchable subsystems $\Theta_0^{(j)}$ from any subsystems of each T_j. As mentioned earlier, the list of arbitrarily switchable locations for the last visited location of an input ACESS can be fetched from this list in polynomial time.
4) Current task specifications Specs $= \{\text{spec}^{(j)} | \text{spec}^{(j)} = \langle h^{(j)}, \rho^{(j)}, c^{(j)}, J^{(j)}, J_{ub}^{(j)}, J_{lb}^{(j)} \rangle$. Here, $h^{(j)}, \rho^{(j)}, c^{(j)}$ are the current sampling period, the current ACESS used, and the execution time for the task T_j. $J^{(j)}, J_{ub}^{(j)}, J_{lb}^{(j)}$ are the current LQR cost, the upper bound and lower bounds for the LQR cost for T_j.
5) The utilization budget U_b depending on the scheduling policy. Any violation of the upper bound of LQR cost directs Algorithm 1 to try and mitigate the violation with a suitable ACESS choice. In such cases, tasks which will be used for relinquishing bandwidth are ones with costs less than the lower bound.

We start by initializing new arrays *utils*, *newUtils* to store the current and desired utilisation of each task (line 1). The task set is then sorted in descending order of their priorities in the shared execution platform (line 2). Starting from the task with the highest priority, for each task, we compute its LQR cost following (3) and store in $J^{(j)}$ (line 5). The utilisation of the jth task is computed using GETUTILS() and stored in *utils* (see line 4). If the control cost of a higher-priority task T_j is beyond its tolerable upper bound $J_{ub}^{(j)}$ (see line 6), Algorithm 1 increases its utilisation by a certain factor. The factor is calculated using the ratio between the actual cost and its given upper bound. The scaled-up utilisation is stored in *newUtils* (see line 7). Since the control cost of this task is beyond the tolerable upper bound, the multiplying factor is always > 1. This ensures the newly assigned utilisation is higher than the current utilisation (both are naturally less than U_b). If doing the above for all control tasks surpasses the total utilisation budget, we do the opposite in the case of the lower-priority tasks (i.e., $(\mathcal{K} - j)$th task from the sorted *TS*, see lines 11–18). If the LQR cost $J^{(j)}$ for a lower-priority task T_j is below its allowable lower bound $J_{lb}^{(j)}$, the utilisation of the task is reduced by a factor of $(J^{(j)}/J_{lb}^{(j)})$ (see line 15). Suppose the total utilisation of the newly assigned task, i.e., SUM(newUtils), is still more than the utilisation budget U_b (line 19). In that case, utilisation for each task is scaled down by a factor of U_b/SUM(newUtils) (see line 21).

In lines 23–29, we derive possible sets of ACESS-s for each task based on the assigned bandwidth utilisation. First, we fetch the list of subsystems for each task, from which we can start in the next HP. We use the GETNEXTLOC() method to compute the arbitrarily switchable subsystems from the last subsystem visited by $\rho^{(j)}$ and store them in $\Theta_0[j]$ (line 24). We fetch the ACESS-s for the tasks assigned with an increased utilisation using the GETACESSMIN() method. As mentioned earlier, given the set of curated ACESS-s for the control task T_j (generated from its CSA) in the form of a list of maps $\mathcal{M}^{(j)}$, the GETACESSMIN() method gives us a set of ACESS-s having a minimum utilisation *newUtils*[j], starting from $\Theta_0[j]$, and with a minimum length same as of $\rho^{(j)}$ (see line 26). They are stored in $\mathcal{M}[j]$ as a map, having the HPs as keys and sorted in ascending order. We fetch the ACESS-s for the tasks that are assigned with reduced utilisation using the GETACESSMAX() method and store them in $\mathcal{M}[j]$ in a similar format (line 27).

The lists of newly computed ACESS-s for each task can have multiple possible HP choices. We find the intersections of the sets of keys in $\mathcal{M}[j]\quad \forall T_j \in TS$ using the method FINDMINCOMMONHP() and store the minimum HP from this common set in *HP* (see line 30). In lines 31–33, we find an ACESS from $\mathcal{M}[j]$ stored against this common HP key. We update the specification of each task specs$^{(j)}$ with this ACESS and its corresponding sampling period in line 32 (remember, we use ACESS-s having fixed sampling period in a single HP) and finally return the updated task specifications *Specs* (in line 34) for deployment in the next HP.

Complexity Analysis: Note that the task-wise utilizations can be calculated using GETUTIL() in $O(1)$. Therefore, the lines 3–10 takes $O(\mathcal{K})$ time. Lines 13–18 again takes $O(\mathcal{K})$ time. As discussed in Section III-F1, the function

GETNEXTLOC() (line 24) runs in $O(1)$ time. Similarly, as discussed in Section III-F1, both GETACESS MIN() (line 26) and GETACESS MAX() (line 26) take $O(\log(m_{\max}))$ time considering a maximum of $m_{\max}$ utilisation entries among all $\mathcal{M}^{(j)}$s for each $T_j \in TS$. Now, consider the minimum and maximum lengths of $\mathcal{M}[j]$s are $m_{j_{\min}}$, $m_{j_{\max}}$, i.e., the ACESS-s are grouped by minimum $m_{j_{\min}}$ and maximum $m_{j_{\max}}$ many HPs. Since they are sorted, the intersection computation method *findMinCommonHP*() among HP key values in $\mathcal{M}[j]$s runs in $O(m_{j_{\min}} \log(m_{j_{\max}}))$ time. Therefore, the overall runtime of Algorithm 1 is $O(\mathcal{K} m_{j_{\min}} \log(m_{j_{\max}}))$, which is much less than $O(\mathcal{K} n_{\max}^2)$ since $m_{j_{\min}} \leq m_{j_{\max}} \leq n_{\max}$ where $n_{\max}$ is the maximum possible subsystem count for each control task (see the complexity calculation in Section III-F1).

TABLE I
IMPLEMENTATION DETAILS AND PARAMETERS SYNTHESIZED FOR CONTROL TASKS

Sys.	Subsystems in CLF $\langle$h(ms) : CSSs$\rangle$	Subsystems in MLF $\langle$ h (ms):CSSs $\rangle$	MDADTs in order (Samples)	Dwell time ratio, GUES, $\mathcal{R}_{safe}$	CAN Data IDs
ESP	$\langle 10: 1, 10, 10^2, 10^3, 10^4, 10^5\rangle$, $\langle 20: 1, 10, 10^2, 10^3\rangle$, $\langle 40: 1, 10, 10^2\rangle$,$\langle 10: 1, 10, 20: 1, 10, 40: 1\rangle$	$\langle 10: 1, 10, 10^2, 10^3, 10^4, 10^5, 10^6\rangle$, $\langle 20: 1, 10, 10^2, 10^3, 10^4, 10^5\rangle$, $\langle 40: 1, 10, 10^2, 10^3, 10^4\rangle$,	2,1,1,1,1,1 1,1,1,1,1 1,1,1,1	1, 0.2, Side Slip $\in [-1, 1]$rad	Rx: 0x111 Tx : 0xA1 (steering ang)
TTC	$\langle 50: 1, 10, 10^2\rangle$,$\langle 50: 1\ 100: 1\rangle$,$\langle 100: 1, 10\rangle$	$\langle 50: 1, 10, 10^2, 10^3\rangle$, $\langle 100: 1, 10, 10^2\rangle$	1,1,1,1 1,1,1	1, 0.9, Trajectory Deviation $\in [-10, 10]$m	Rx: 0x121 Tx : 0xC4 (acceleration)
CC	$\langle 10: 1, 10, 10^2\rangle$,$\langle 10: 1\ 20: 1\rangle$,$\langle 20: 1, 10\rangle$	$\langle 10: 1, 10, 10^2\rangle$, $\langle 20: 1, 10\rangle$	2,1,1 2,1	2, 0.8, Velocity $\in [-5, 5]$m	Rx :0x131 Tx : 0xD1 (throttle Ang)
SC	$\langle 20: 1, 10, 10^2, 10^3\rangle$ $\langle 20: 1, 10, 40: 1, 60: 1\rangle$,$\langle 40: 1, 10,, 10^2, 10^3\rangle$,$\langle 60: 1, 10, 10^2\rangle$	$\langle 20: 1, 10, 10^2, 10^3, 10^4\rangle$, $\langle 40: 1, 10, 10^2, 10^3\rangle$, $\langle 60: 1, 10, 10^2\rangle$	1,1,1,1,1 1,1,1,1 1,1,1	1,0.8, Car Position $\in [-0.1, 0.1]$m	Rx: 0x141 Tx : 0xF4 (force)

IV. EXPERIMENTAL RESULTS

Experimental Setup: We use two resource-constrained safety-critical CPSs, automotive and quadcopter, as our case studies. For both case studies, we use the LQR-based optimal control technique. The respective control tasks, along with several other tasks, are implemented in an ARM-based 32-bit Infineon Aurix TC-397 ECU running at 300 MHz. These tasks, along with Algorithm 1, are scheduled for execution in the same core of this ECU following a fixed-priority schedule. Algorithm 1 runs with the lowest priority once in every HP. It updates the control task schedules in the subsequent HPs. The running time of Algorithm 1 is found to be $<$ 1ms in both case studies. We build a 500-Kb/s controller area network (CAN) setup that connects this ECU with an ETAS Labcar Real-time PC, where we emulate the physical system dynamics.

Automotive Case Study: We implement four automotive control tasks in the electronic stability program (ECU): (ESP, maintains yaw stability), trajectory tracking control (TTC, regulates deviation from a desired longitudinal trajectory), cruise control (CC, maintains a desired vehicle speed), and suspension control (SC, manages vehicle suspension in different road/driving conditions) [6], [18] (refer to Table I Col.1). The performance requirements for these controllers (as mentioned in Col. 5 of Table I) need to be achieved using the limited processing and communication bandwidths. This makes automotive embedded systems an ideal case study for our performance-aware bandwidth-sharing solution. Following the AUTOSAR mandates [19], we implement separate reception tasks that filter and receive sensor IDs transmitted by the Labcar RTPC in CAN. On asynchronous updation of task-specific sensor data labels, the control tasks are run, followed by their corresponding transmission tasks to transmit the computed control data through CAN for plant actuation.

CSA Synthesis: We start by deriving the switching parameters for each of the control loops as discussed in Section III-D to synthesize their CSAs for the desired GUES stability criteria as provided in Col.5 of Table I. We use a random simulation-based verification method to choose subsystems for each control loop that always keep the system outputs within a given safe operating region $\mathcal{R}_{\text{safe}}$ as mentioned in Col.5. While choosing this set of controllable sampling periods and the maximum consecutive skips, we consider a delay margin that arises during actuation as observed in our setup. We use YALMIP with the Mosek optimization engine to solve the LMIs given in (18) to find out the switchable subset of subsystems with MLFs or CLFs. As a result of these computations for every control loop, in Col.2 and Col.3, we provide the switchable subsystems in terms of periodicity-CSS combinations. In Col.2, the subsystems that are confined within an angle bracket have a CLF and can be arbitrarily switched. On the other hand, in Col.3, the subsystems confined within an angle bracket have MLFs and can slowly switch between themselves by maintaining certain MDADTs, as mentioned in Col.4. The MDADT corresponding to a subsystem is given in the same order as the subsystem appears in the tuple (angle brackets), and it is derived in terms of the number of sampling periods. The *dwell time ratios* corresponding to each control task are provided in Col.5; however, the safe choice of the subsystems disallows any unstable subsystems in the switchable set.

Using these subsystems and their corresponding parameters, we can synthesize CSAs corresponding to each control loop and optimally store switchable ACESS-s as mentioned in Section III-F1. In Fig. 6, we demonstrate the behavior of such safe subsystem choices for suspension control. Each subfigure in Fig. 6 plots the car position (in meters) on the y-axis and time (in terms of sampling period count) on the x-axis. The blue plots denote output characteristics under different periodic control executions and the red circled plots present the system characteristics under skipped executions. Note that as we increase the control execution frequency (from 20 ms in the leftmost to 120 ms in the rightmost plot), the system stabilizes faster. On the other hand, as we increase the number of consecutively skipped executions after the actuation/control execution at 20 ms, the system shows some undershoot but remains within the safe region, i.e., $[-0.1, 0.1]$ (see Table I, Col.5, Row.4). However, note that among these controllers with multiple sampling periods, only the controllers with 20 ms, 40 ms and 60 ms and only the following CSSs, $1_{0.02}$, $1_{0.02}0_{0.02}$, $1_{0.02}(0_{0.02})^2$, $1_{0.02}(0_{0.02})^3$ have CLFs given our performance criteria.

Dynamic Scheduling and Comparison With SOTA: We demonstrate how Algorithm 1 operates for the automotive control tasks, with an example scenario in Fig. 7. In all four subfigures in Fig. 7, we plot *system outputs in blue* (car position for SC and side slip for ESP) on the left y-axis and *the LQR control costs with dashed gray line* on the

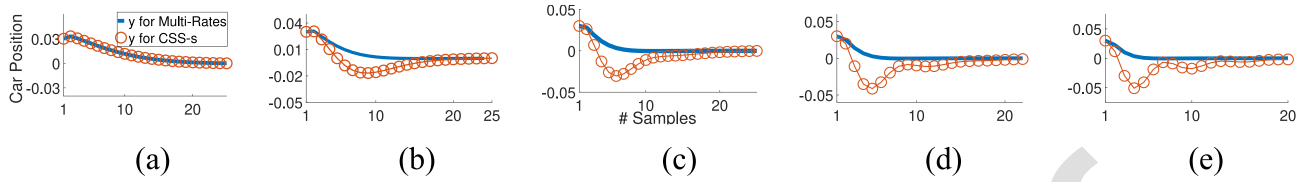

Fig. 6. CSSs and Periodicity changes for suspension control. (a) $h = 0.02$ and CSS = 1. (b) $h = 0.04$ and CSS = 10. (c) $h = 0.08$ and CSS = 100. (d) $h = 0.1$ and CSS = 1000. (e) $h = 0.12$ and CSS = 10 000.

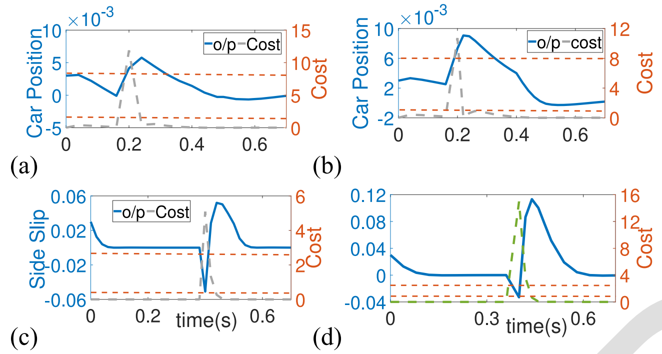

Fig. 7. Comparison between proposed method and SOTA. (a) SC under our dynamic scheduling. (b) SC under skipped execution only. (c) ESP under our dynamic schedule. (d) ESP under only periodicity change.

right y-axis, w.r.t time (in seconds) in the x-axis. In Fig. 7(a) and (c) (left subfigures), we demonstrate system outputs under the proposed dynamic scheduling and in Fig. 7(b) and (d) (right subfigures) we demonstrate the effect of SOTA, i.e., only multirate scheduling [16], [20] and only control execution skip-based scheduling [6], [21] approaches. For our experiments, we consider the nominal control cost observed during steady state as the cost lower bound for each control task. The cost observed during transient states is assumed as allowable upper bounds.

At the start of the scenario, SC is running with the controller for a 40 ms sampling period and with ACESS 10101 (utilisation = 3%). Due to bad road conditions, there is a sudden change in the car position, and normalized control cost for SC increases beyond the tolerable upper bound of 8 [marked with orange dashed lines in top subfigures, Fig. 7(a) and (b)]. In this situation, the online Algorithm 1 assigns more (4%) utilisation by deploying the 20 ms controller with ACESS 1011111101 for SC, whereas the SOTA strategies that only rely on the skipped control executions (no periodicity change) assign 5% utilisation by deploying an ACESS-s $(1_{0.04})^5$ (i.e., with the same sampling period of 40 ms). As can be seen, with our approach, the output settles faster [cf. the settling of the blue plot in Fig. 7(a) compared to Fig. 7(b)].

The ESP task, on the other hand, runs with a controller of 20 ms sampling period and with ACESS 1010101010 (utilisation 2.5%) till 0.02 ms following our strategy. Notice that its control cost is below the lower bound [orange dashed line at 0.1 on the right axis in Fig. 7(c) and (d)]. Hence, to provide more utilisation to SC, Algorithm 1 assigns ACESS 1000110010 (utilisation 2%) to ESP and continues the same till 0.04 s, whereas under the SOTA strategies that solely rely on multirate scheduling (no control skips), ESP runs with a controller of 40 ms sampling period from the start and cannot reduce the utilisation since there is no controller with any higher-sampling period that is arbitrarily switchable from the current 40 ms controller and still operates within the safe region (see Table I Col.2, row.1). Due to sudden arrival/discovery of an obstacle at 0.04 s, there is a sudden deviation in side slip angle causing a high-control cost for ESP beyond the tolerable upper bound of 2.5 (ref. the orange dashed lines in bottom subfigures). Our methodology deploys ACESS $(1_{0.02})^{12}$ (i.e., 111111111111 ACESS under 20 ms sampling period, with 5% utilisation), and the SOTA multirate strategy switches to 20 ms controller (as 40 ms and 20 ms controllers have CLF respecting given GUES, see Table I) assigning the same 5% utilisation. But as can be seen in Fig. 7(d), the switching of the periods causes a larger overshoot, in turn causing *doubled* control cost compared to our strategy [Fig. 7(c)].

Quadcopter Case Study: There are three control tasks in our quadcopter case study [22], namely, 1) *altitude control* (*AltCon*, maintains a desired height); 2) *TTC* (*TTC*, tracks a desired x-position); and 3) *quadcopter stability control* (*QSC*, tracks a desired y-position). They have higher frequencies than automotive control tasks from the previous case study and consume 50% of the overall processing bandwidth. Other tasks include transmission, reception, and dummy signal (image) processing tasks that consume 38% of the bandwidth. The desired GUES decay rate $\gamma = -0.2$ and safe operating regions $\mathcal{R}_{\text{safe}}$ spanning [2, 20], [−2, 2], and [−5, 5] m around the references of *AltCon*, *TTC* and *QSC*, respectively, are input to our proposed methodology. The ACCESS-s generated from the synthesized CSAs for the provided inputs are task-wise stored offline. The implementations and test cases can be checked at https://github.com/SunandanAdhikary/DynamicSchedulingCSAs. We evaluate our dynamic scheduling algorithm and SOTA methods in the following two scenarios of a flight trajectory spanning 70 s. *Scenario 1: AltCon* faces an external disturbance that deviates the quadcopter from its desired altitude 10 m during 15–18 s; *Scenario 2:* later, during 25–42 s *QSC* faces multiple obstacles in its path and changes its desired references from 15 to −20 m. Figs. 8 and 9 plot the plant responses (on the left axis in blue) and control costs (on the right axis in dashed gray) in these scenarios under the proposed methodology and SOTA multirate scheduling strategies, respectively. Initially *AltCon*, *TTC*, *QSC* follow ACESS-s that utilize 16%, 11%, and 20% of the processing bandwidth, respectively.

During Scenario 1, SOTA control skipping policies deploy the same ACESS as Algorithm 1 does to *AtlCon*. This

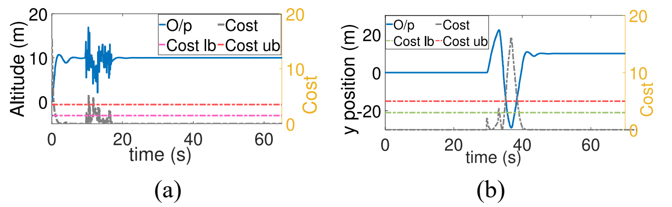

Fig. 8. Dynamic scheduling in quadcopter with our approach. (a) Altitude control: Under noise. (b) QSC: Avoiding obstacles.

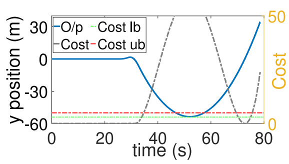

Fig. 9. QSC: Multirate.

deployed ACESS utilizes 20% of the total processing bandwidth, which is more than *AltCon*'s initial bandwidth assignment. As can be seen in Fig. 8(a), following the deployed ACESS with higher utilisation, *AltCon* successfully stabilizes the quadcopter at the desired altitude of 10 m even in the presence of disturbance (before 20 s). To keep the overall bandwidth within budget, Algorithm 1 releases bandwidth from *QSC* as it has the lowest priority and its cost deviation during Scenario 1 remains below the allowable bound (marked with a dashed green line). It allocates an ACESS with an increased periodicity, which only utilizes 9% of the bandwidth, whereas the ACESS deployed by the SOTA control skipping techniques for *QSC* during this Scenario 1 consumes 13% (i.e., 4% more) processing bandwidth with the same performance cost. This happens because Algorithm 1 being less conservative than SOTA, discovers more ACESS choices.

During Scenario 2, using SOTA multirate scheduling strategy makes *QSC* unstable due to frequent switching between multiple periodicities. Notice in Fig. 9 that the y-position becomes unstable and leads to unsafe behavior, causing the cost to increase beyond the allowable upper bound (red dashed line). As can be seen in Fig. 8(b), in the same scenario, Algorithm 1 deploys suitable ACESS with the same periodicity having a 16% bandwidth utilisation to minimize this cost deviation. This newly allocated bandwidth is higher than the bandwidth allocated to *QSC* during Scenario 1 but respects the utilisation budget. By using this ACESS with increased utilisation, *QSC* successfully avoids obstacles and attains its desired y-position before 42ms, minimizing its control cost.

V. Conclusion

We provide a framework for subsystem identification and finitary representation of switching constraints by considering under a common umbrella the existing paradigms of multirate as well as weakly hard scheduling of control tasks. This is then leveraged to efficiently schedule control loops on resource-constrained shared platforms. In the future, we intend to use this automata-theoretic representation for control-scheduling co-designs in 1) nonlinear hybrid systems and 2) more complex multicore platform mappings.

References

[1] A. Cervin, M. Velasco, P. Martí, and A. Camacho, "Optimal online sampling period assignment: Theory and experiments," *IEEE Trans. Control Syst. Technol.*, vol. 19, no. 4, pp. 902–910, Jul. 2010.

[2] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE Trans. Comput.*, vol. 50, no. 4, pp. 308–321, Apr. 2001.

[3] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Trans. Comput.*, vol. 44, no. 12, pp. 1443–1451, Dec. 1995.

[4] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: Deadline-miss-aware control," in *Proc. 31st Euromicro Conf. Real-Time Syst.*, 2019, pp. 1–24.

[5] S. Linsenmayer and F. Allgower, "Stabilization of networked control systems with weakly hard real-time dropout description," in *Proc. IEEE 56th Annu. Conf. Decision Control (CDC)*, 2017, pp. 4765–4770.

[6] S. Ghosh, S. Dutta, S. Dey, and P. Dasgupta, "A structured methodology for pattern based adaptive scheduling in embedded control," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 1–22, 2017.

[7] H. Liang, Z. Wang, R. Jiao, and Q. Zhu, "Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees," in *Proc. 39th Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.

[8] S. Xu, B. Ghosh, C. Hobbs, P. Thiagarajan, and S. Chakraborty, "Safety-aware flexible schedule synthesis for cyber-physical systems using weakly-hard constraints," in *Proc. 28th Asia South Pacific Design Autom. Conf.*, 2023, pp. 46–51.

[9] N. Vreman, P. Pazzaglia, V. Magron, J. Wang, and M. Maggio, "Stability of linear systems under extended weakly-hard constraints," *IEEE Control Syst. Lett.*, vol. 6, pp. 2900–2905, 2022.

[10] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. Di Natale, "Beyond the weakly hard model: Measuring the performance cost of deadline misses," in *Proc. 30th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2018, pp. 1–22.

[11] P. Pazzaglia, A. Hamann, D. Ziegenbein, and M. Maggio, "Adaptive design of real-time control systems subject to sporadic overruns," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2021, pp. 1887–1892.

[12] S. K. Ghosh, S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty, "Design and validation of fault-tolerant embedded controllers," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2018, pp. 1283–1288.

[13] S. Ghosh, S. Dey, and P. Dasgupta, "Performance-driven post-processing of control loop execution schedules," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 26, no. 2, pp. 1–27, 2020.

[14] D. Soudbakhsh, L. T. Phan, O. Sokolsky, I. Lee, and A. Annaswamy, "Co-design of control and platform with dropped signals," in *Proc. ACM/IEEE 4th Int. Conf. Cyber-Phys. Syst.*, 2013, pp. 129–140.

[15] X. Zhao, L. Zhang, P. Shi, and M. Liu, "Stability and Stabilization of switched linear systems with mode-dependent average dwell time," *IEEE Trans. Autom. Control*, vol. 57, no. 7, pp. 1809–1815, Jul. 2012.

[16] M. Schinkel, W.-H. Chen, and A. Rantzer, "Optimal control for systems with varying sampling rate," in *Proc. Amer. Control Conf.*, vol. 4, 2002, pp. 2979–2984.

[17] X. Du, C. Ramakrishnan, and S. A. Smolka, "Tabled resolution+ constraints: A recipe for model checking real-time systems," in *Proc. 21st IEEE Real-Time Syst. Symp.*, 2000, pp. 175–184.

[18] S. Adhikary, I. Koley, S. Maity, and S. Dey, "Work-in-progress: Control skipping sequence synthesis to counter schedule-based attacks," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, 2022, pp. 491–494.

[19] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proc. 6th Int. Workshop Anal. Tools Methodol. Embed. Real-time Syst. (WATERS)*, vol. 130, 2015, pp. 1–6.

[20] P. Marti, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes, "Optimal state feedback based resource allocation for resource-constrained control tasks," in *Proc. 25th IEEE Int. Real-Time Syst. Symp.*, 2004, pp. 161–172.

[21] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-system stability under consecutive deadline misses constraints," in *Proc. 32nd Euromicro Conf. Real-Time Syst. (ECRTS)*, 2020, pp. 1–4.

[22] P. Wang, Z. Man, Z. Cao, J. Zheng, and Y. Zhao, "Dynamics modelling and linear control of quadcopter," in *Proc. Int. Conf. Adv. Mechatron. Syst. (ICAMechS)*, 2016, pp. 498–503.