# Revisiting Dynamic Scheduling of Control Tasks: A Performance-aware Fine-grained Approach

***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 minimsed. 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.**

## I. Introduction

With more number of autonomous features and adaptive control modes available in modern-day cyber-physical systems (CPS), the corresponding task sets to be executed in their electronic control units (ECUs) have also increased significantly. Due to their safety-critical nature, such real-time systems are designed with safety and timeliness as primary design criteria. Enforcing these requirements requires correctness in terms of both control design and platform-level schedulability of control tasks. In order to efficiently utilise the available processing and communication bandwidth in resource-constrained architectures typically employed as CPS platforms, new standards for adaptive control and platform scheduling support have emerged in the last few years, for example, Adaptive Autosar in the automotive domain [1].

Significant research has happened in this domain of dynamic scheduling of control tasks where, depending on run-time performance requirements, the task period is dynamically switched while satisfying stability and baseline performance [2]–[5]. Such adaptiveness can help in addressing a more demanding scenario faced by one controller by increasing its frequency using bandwidth relinquished by some other controller for which a decrease in frequency may be admissible at that time. Such techniques help in gleaning out better performance over statically scheduled control tasks with a nominal choice of periodicities.

However, such *multi-rate* 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 [6]. *Weakly-hard* constraints of a closed loop system can be captured as $(m, k)$-firm specifications [7], i.e. maximum $(k-m)$ deadline misses or *control execution/actuation skips* are allowed in every $k$ consecutive control task instances. Such works build on constraint identification on packet dropout rate or drop rate for control tasks as was first investigated by the authors in [8] to maintain performance in networked control systems (NCS). The reason behind such resilience, as exhibited by controller implementations can be explained as follows. To achieve robustness, the sampling frequencies of feedback controllers are usually kept much higher than the maximum frequency component of the open-loop system. This facilitates the attainment of desired control performance even under a bounded amount of actuation delays due to deadline misses of control task executions [9]. Such relaxation of constraints on control loops usually helps in budgeting slack for unforeseen timing uncertainties often introduced due to unmodeled dynamics, platform level faults, jitters, task execution overrun, communication delays etc in the complex network components and embedded processing units. Researchers have demonstrated how such weakly-hard constraints can be leveraged to co-schedule control tasks in a more efficient way [10]. Such works advocate that every control loop executes a cyclic job schedule while following some weakly hard constraint at some fixed sampling period. In a dynamic scenario, when some task needs to execute more often, i.e. skip less number of job instances, some other control task relinquishes processor bandwidth by skipping some of its scheduled execution while satisfying weakly-hard constraints. In the present work, we aim to combine techniques from both multi-rate and weakly-hard constraint-based task executions together and examine possible advantages of such a hybrid scheduling approach in case of control architecture co-design.

**Related work:** One of the initial works done on an online period assignment of control tasks while maintaining their performance can be found in [2]. This work introduces the idea of a control scheduling co-design-based framework that motivates an integrated control and scheduling policy design in a resource-aware manner and provides a survey of prior works that motivated such ideas. The authors of [3] proposed an optimal online period assignment method by analysing the cumulative control cost incurred by a schedulable set of periodicities for control tasks. For this, it uses a feedback-based scheduler that accounts for the noise intensity and resource utilisation to assign periodicity. A similar method has also been developed for delay-aware controllers in [4],

where the authors thoroughly analyse the controller response times and use these delay estimates to assign task periodicities within a utilisation budget under a fixed priority scheduler. However, such switching decisions must be system-specific; otherwise, amplification of stabilizing feedback control gains to compensate for longer sampling durations may cause undesired transient behaviour leading to instability. The work done in [11] proposes a methodology for sampling period switching, that avoids such unstable switching sequences. Instead of control cost optimization, this work constrains the set of periodicities for a multi-rate control system to maintain Lyapunov stability. In [12], the authors propose a cost-optimized online period assignment using dynamic priority-based scheduling (resource-based earliest deadline scheduling or RBED). Their proposed methodology promises better resource utilisation and control cost optimization compared to other state-of-the-art (SOTA) works. Authors in [13] proposed an online algorithm that promises lower compute overhead compared to other online control cost optimization algorithms that assign periodicities under resource utilisation budget. However, due to stability and utilisation constraints, such online periodicity switching policies are not very flexible and are left with a handful of choices. This motivates the idea of a weakly hard design paradigm where a bounded number of control execution skipping is encouraged based on resource availability and performance requirements of the tasks.

In this line of work, authors propose derivation of stable weakly hard control execution schedules for multiple tasks implemented in an embedded control unit [6], [9], [14], [15]. Similar concepts have been leveraged to optimally allocate processing bandwidth depending on the system states in [9], [16]. The authors in [17] synthesize fault-tolerant weakly hard controller specifications and evaluate them for a realistic automotive setup, whereas others mainly demonstrate synthetic case studies. Works done by the authors of [15] look into the problem from a switched system perspective, where the system under control execution/actuation and skipping of control execution/actuation are considered to be 2 modes to switch between. These works formulate an automaton to accept well-performing switching sequences between these 2 modes based on their Quality of Service (QoS) or performance. In similar lines, the authors of [18] propose different control execution skipping strategies that can be chosen depending on the system performance and available resources in embedded platforms to achieve optimal performance and resource consumption. Most of these works are done by analysing *joint spectral radius* [18] for a weakly-hard specification (i.e., maximal asymptotic growth for all allowed subsystem switching sequences) and how it abides by a *Common Lyapunov Function (CLF)*, designed for a desired performance specification. However, these works are mostly evaluated for standalone or synthetic task sets. The work in [19] aptly analyzes the maximum number of allowable control execution skips in an embedded platform following a similar fashion to the aforementioned works but using *Multiple Lyapunov Function (MLF)*. For a given performance specification, switching strategies designed using MLFs are less restrictive compared to CLF-based strategies as they allow slower switching between a larger number of modes/subsystems [20], [21]. MLF mandates a switched system to *dwell* in different subsystems for different minimum durations based on their performances. This minimum duration is known as *Mode-dependent Average Dwell Time (MDADT)* [20], and unlike the popularly used Average Dwell Time (ADT), it is not subsystem-agnostic [21]. In the following section, we explain our novel strategy that combines MDADT and MLF-based switching with the state-of-the-art (SOTA) approaches for dynamic task scheduling.

**Novelty and Contribution:** All of the aforementioned works either assign periodicities to control tasks such that control cost is minimized or decide weakly hard specifications for them such that their performances remain uncompromised as they become schedulable in the execution platform. We strategically utilise both these disciplines in order to minimize state degradation by resharing processing bandwidth as per the requirement of every task. The aperiodic activation patterns for a discrete control loop are selected by observing its current system state degradation, such that a closed loop with higher state degradation has a higher number of control execution instances, i.e., it is executed with less number of skipped instances or with higher frequency. While switching between different periodicities, we ensure the control loops discretized for those periodicities have a *Common Lyapunov Function (CLF)* similar to the SOTA approaches. We integrate this with a *Multiple Lyapunov Function (MLF)*-based stability analysis of control execution skipping sequences that relaxes the control execution skipping limits and employs an *MDADT*-based slow switching technique. Combining state-of-the-art and novel techniques, our methodology formulates a more fine-grained switching between aperiodic activation patterns that are safe and performance-aware for the control tasks scheduled in a shared embedded platform. To summarize, the following are the contributions of this work.

**1.** For each participating control loop, we work out an MDADT-based switching strategy for safe switching between the choice of periodicity as well as the choice of control execution skips. This allows more relaxed switching with comparatively less bandwidth than the SOTA. We theoretically ensure that this switching strategy ensures a given stability margin without hampering safety.

**2.** We synthesize a *Control Skipping Automaton or CSA*, which utilises the theoretically derived stable switching strategies to switch between the subsystems with different control loop skipping capabilities. For a control loop with fixed periodicity, given its required number of task executions within a hyper-period (i.e., weakly hard performance specification) along with its safety and performance criteria (eg., forward invariant region, exponential stability criteria, etc.), this finite state automaton generates stable aperiodic activation patterns for the control tasks.

**3.** We devise an algorithmic strategy to dynamically deploy periodic and aperiodic activation patterns for different control loops running in a shared embedded platform. For each control task, given the current performance degradation, safety, and performance specifications, the algorithm suitably assigns schedule activation patterns by leveraging the respective *CSA*-s while respecting the processor utilisation budget.

**4.** We evaluate our performance-aware dynamic resource-sharing technique in a practical automotive setup and compare it with SOTA approaches to analyze its effectiveness.

## II. System Model

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

$$\dot{x}(t) = \Phi x(t) + \Gamma u(t) + w(t),\ y(t) = Cx(t) + v(t) \quad (1)$$
$$\hat{x}[k+1] = A\hat{x}[k] + Bu[k] + L(y[k] - C\hat{x}[k]),\ u[k] = -K\hat{x}[k]$$

Here the vectors $x \in \mathbb{R}^n$, $\hat{x} \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, and $u \in \mathbb{R}^p$ define the plant state, estimated plant state, output, and control input, respectively. $\hat{x}[k], y[k], u[k]$ are their values at $k$-th ($k \in \mathbb{N}$) sampling instance. At the $k$-th sampling instance, the real-time is $t = kh$, where $h$ is the sampling period. $w(t) \sim \mathcal{N}(0, Q_w)$, $v(t) \sim \mathcal{N}(0, R_v)$ are process and measurement noises. They follow Gaussian white noise distributions with variances $Q_w \in \mathbb{R}^{n\times n}$ and $R_v \in \mathbb{R}^{m\times m}$, respectively. The matrices $\Phi, \Gamma$ are the continuous-time state and input-to-state transition matrices, respectively. The discrete-time counterparts of these matrices are, $A = e^{\Phi h}$, $B = \int_0^h e^{\Phi t}\Gamma dt$. $C$ is the output transition matrix. The feedback controller module samples the plant output $y[k]$ periodically once in every $h$ duration and estimates the plant state $\hat{x}[k]$ using a Luenberger observer with Kalman Gain $L$ to filter the Gaussian noises. The control input $u[k]$ is calculated from this estimated state using $K$ feedback gain. The control input $u[k]$ is communicated via the communication medium that connects the plant and control unit and is applied to the plant through the actuators. The feedback control gain $K$ is designed as per the performance requirements, as discussed in the following section.

*1) Control Design, Performance Metrics & Safety Criteria:* A control design metric represents the control objective while designing the controller. One such design metric that we often use 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. On the other side, the *control performance* is the measure of the quality of control (QoC), i.e., how efficiently the design requirement is met. In this work, we consider a Linear Quadratic Regulator (LQR)-based controller design technique. So we use the ***LQR cost function*** as the performance metric.

$$J = \sum_{k=0}^{N-1} (x[k] - r[k])^{\mathsf{T}} Q (x[k] - r[k]) + u^{\mathsf{T}}[k] R u[k]) \quad (2)$$
$$+ (x[N] - r[N])^{\mathsf{T}} S (x[N] - r[N])$$

This denotes the finite horizon control cost for a system [22], with symmetric weighing matrices $Q, R \succ 0$ capturing the relative importance that the control designer can give to the state deviation and control effort, respectively. $S$ is the final state cost matrix. Minimizing the control cost improves the control performance. Replacing $u[k]$ with $-Kx[k]$ we derive the optimal feedback control gain $K$ that minimizes control cost $J$. This is done by solving a discrete-time Riccati equation [22]. In this work, we consider exponential stability as a performance-driven stability metric. Given a settling time requirement, we can express it in terms of a desired minimum exponential decay within the mentioned settling time duration so that the system output stays within a $2\%$ error bound of a desired reference. Theoretically, we express it using the notion of the Global Uniform Exponential Stability (GUES) criterion as defined below.

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

We tune the state cost matrix $Q$ to design an LQR gain $K$ that promises a closed-loop decay similar to the desired GUES [22].

The phase difference between the input and output of a control system can cause *unsafe* transient behaviour even in the stable control loops (eg., overshoots, undershoots). Our definition of *safety* demands the system states to initialize and always remain within a forward invariant *safe operating region* $\mathcal{R}_{safe}$ such that $x(t) \in \mathcal{R}_{safe} \Rightarrow \forall t' \geq t,\ x(t') \in \mathcal{R}_{safe}$.

*2) Closed-loop under Sampling Period Change:* In the case of multiple control loops using the same embedded platform, the designers assign a fixed sampling period to each of the closed loops in order to make the control tasks schedulable in the embedded platform. Researchers also have proposed dynamic periodicity assignments to optimize the control cost [11]. Notice that the discrete-time system characteristics matrices depend on their sampling period. Therefore, we express them as a function of sampling period $h$, i.e., $A(h) = e^{\Phi h},\ B(h) = \int_0^h e^{\Phi t}\Gamma dt$. Accordingly, the LQR and Kalman gains also change, i.e., $K(h)$ and $L(h)$, respectively. By discretizing the continuous-time plant state equation given in Eq. 1 for $h$ sampling period, we get $x[k+1] = A(h)x[k] + B(h)u[k]$. We intend to capture both the plant and controller states in 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 $Cx$, the overall closed-loop system evolves as follows (Since all are for the same sampling period $h$, we omit $h$ variable from RHS):

$$X[k+1] = A_{1,h}X[k],\ A_{1,h} = \begin{bmatrix} A & BK \\ LC & A - LC + BK \end{bmatrix} \quad (3)$$

For multiple sampling periods, $h_i, h_j$ say, we denote the discrete-time augmented system matrices as $A_{1,h_i}, A_{1,h_j}$. In all cases, we choose systems with *safe transient behaviours*, i.e., the designed feedback gain produces no undershoot/overshoot beyond $\mathcal{R}_{Safe}$. Each of these closed loops can be considered as a separate sub-system denoted as $\phi_{h_i}$ and $\phi_{h_j}$ respectively (evolving following Eq. 3), with the characteristic matrices $A_{1,h_i}$ and $A_{1,h_j}$. Note that when a stable augmented system changes its sampling period with a new choice of stabilizing LQR and Kalman gains designed for the new sampling period, unstable transient behaviour may exist [11]. For this, we need to constrain the switching to certain sampling

period choices, which we discuss in Sec. III-C. Before that, let us discuss how such an augmented system behaves in the presence of aperiodic control executions/actuations.

*3) Closed-loop under Skipped Control Executions:* As described earlier, the feedback control input of a closed-loop needs to be periodically computed based on current system states and actuated within a predefined system sampling period. Even after assigning proper sampling periods to schedule multiple control task executions or communications in shared processing or communication medium, issues may arise. In the case of networked or embedded control systems, there are *network packet drops* while transmitting control updates or *deadline misses* while executing control tasks. This causes the ideally periodic control executions to become *aperiodic*. In case of such aperiodic control executions/actuation, the actuator on the plant side does not receive any new control update within a sampling interval $[k, k+1)$ or time interval $[kh, kh+1)$. So, the value of the control input remains the same as it was in the last iteration (last received control actuation) i.e., $u[k+1] = u[k]$. Leveraging the robustness available in control loop design accounting for potential actuation misses, we 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. 1 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 iteration, the closed-loop system in Eq. 1 evolves as $x[k+1] = A(h)x[k] + B(h)u[k]$, $\hat{x}[k+1] = I\hat{x}[k] + Ou[k] + Oy[k]$, $u[k+1] = K(h)\hat{x}[k+1]$ . $I$ 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 (To save space, we omit variable $h$ in RHS).

$$X[k+1] = A_{0,h}\ X[k],\ A_{0,h} = \begin{bmatrix} A & BK \\ O & I \end{bmatrix} \tag{4}$$

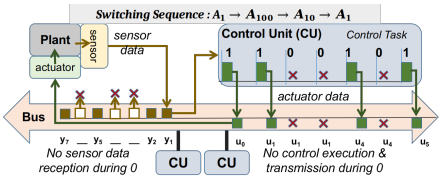



Figure 1: Closed-loop with Skipped Control

To judge the system performance under control execution/actuation skips, we need to model the system as a discrete-time system (sampling period $h$) switching between the closed-loop augmented system characteristic matrices $A_{1,h}$ from Eq. 3 and $A_{0,h}$ from Eq. 4. Notice that since the sampling period is the same, the feedback control gain and Kalman gain remain the same for both. Such a system can be represented as a switched LTI system where it switches between different combinations of $\{A_{1,h}, A_{0,h}\}$. An example is presented in Fig. 1. Here, after executing the control task at $k$-th and $(k+1)$-th iteration, we skip the control executions at $(k+2)$-nd and $(k+3)$-rd sampling iterations, then the augmented closed-loop system state at $(k+4)$-th iteration becomes $X[k+4] = A_{0,h}A_{0,h}A_{1,h}A_{1,h}X[k]$. So, we are using the control action 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 action is computed at $t = (k+3)h$ and used for the next sampling period, i.e., for $t \in [(k+3)h, (k+4)h)$. We express this switching control sequence as *Aperiodic Control Execution Skipping Sequence (ACESS)*.

**Definition 2 (Aperiodic Control Execution Skipping Sequence).** *An $l \in \mathbb{N}$ length aperiodic control execution skipping sequence (ACESS) for a given control loop is a sequence $\rho \in \{0_{h1}, 1_{h1}, 0_{h2}, 1_{h2}, \cdots, 0_{hM}, 1_{hM}\}^l$ such that $\{h1, h2, \cdots hM\}$ are the sampling periods of its available set of feedback controllers where $\forall h \in \{h1, h2, \cdots hM\}$, $1_h$ denotes a control execution when the augmented closed-loop system state $X$ progresses for $h$ time duration following Eq. 3, and $0_h$ denotes a skipped execution when $X$ progresses $h$ duration following Eq. 4.* □

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 system state is periodically feedback-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

As discussed earlier, our methodology employs both *periodic* and *aperiodic* activations to the control tasks depending on their performance degradations. In this section, we present our methodology, starting with a brief overview.

### A. Overview

Initially, the control tasks are assumed to be co-scheduled with certain periodicity choices in the shared platform. The proposed methodology is designed to observe a *state degradation metric*, e.g., LQR control cost (see Eq. 2), for every control task once in every *hyper-period* (the time duration after which the task execution pattern repeats) to understand how much it is affected by noise. When the state degradation metric is beyond a *safe threshold*, our methodology assigns *higher* execution bandwidth by increasing the number of control executions/actuation either by updating the control execution skipping sequence with less number of skips or by reducing the periodicity (i.e., increasing the frequency of control actuation/execution) of the control task. If the processor utilisation goes beyond a fixed budget due to these higher bandwidth assignments, we need to reduce the bandwidth consumption of other tasks having steadier control output or minimum performance deviation. This is again done by either skipping more execution instances of such control tasks or by increasing their periodicities (i.e., decreasing the frequency of control

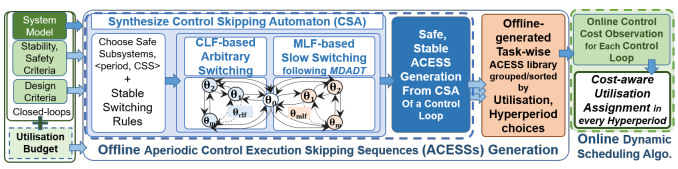



Figure 2: Overview of the proposed framework

actuation/execution). Doing the same requires two important components.

• A library of possible ACESS 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.

• A set of possible constraints that ensure stable switching across sampling periods and across ACESS-s.

As shown in Fig. 2, 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. We next elaborate on the method for creating such time-constrained switching sequences.

### *B. Aperiodic Control Executions: A 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 (or delayed actuation for certain sampling periods), during which it uses the last actuated control signal. We consider this continuous aperiodic evolution of the augmented closed-loop as a *Control Skipping Sub-sequence (CSS)*. The subsystems are chosen such that their time and frequency domain characteristics always ensure safe transient behaviour (i.e., by confining system states within a safe region, without any over/undershoots).

**Definition 3** (**Control Skipping Sub-sequence).** *A control skipping sub-sequence in an $l$-length ACESS $\rho$ is an $i(\leq l)$-length sub-string having the form $1_h(0_h)^{i-1}$ such that here also, $1_h$ signifies a control execution/actuation and augmented state progression using computed control input for $1$ sampling period or $h$ duration following Eq. 3. The following $0$s signify the actuation of that control input once in each of the next $(i-1)$ sampling iterations ($(i-1)h$ time) such that the states evolve following Eq. 4 at each of those iterations.* □

**Example 1.** *The $l = 7$ length ACESS $\rho = 1_{hi}1_{hi}0_{hi}0_{hi}1_{hj}0_{hj}1_{hj}$ spans for $7$ sampling iterations, i.e., during $t \in [khi, (k+4)hi + 3hj)$ has 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 $A_{100,hi} = A_{0,hi}^2 A_{1,hi}$, $1_{hj}0_{hj}$ at $\rho[4], \rho[5]$, signifying a control execution and state evolution using $A_{10,hj} = A_{0,hj}^1 A_{1,hj}$ and $1_{hj}$ at $\rho[6]$ signifying a control execution and state evolution using $A_{1,hj}$. Note that, corresponding to each such $i$-length CSS, $1_h 0_h^{i-1}$, the controller executes once followed by the continuous evolution of the plant over a time window of $i \times h$ instead of $h$ (sampling period of the controller), 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., $\rho$ contains total $N$ control skipping sub-sequences), a closed-loop dynamical system, discretized with a set of sampling periods $\{h1, h2 \cdots hM\}$, behaves like a switched system having state evolution of the form: $x[l] = (A_{0,hM})^{i_N-1}A_{1,hM}\ldots(A_{0,h1})^{i_2-1}A_{1,h1}(A_{0,h1})^{i_1-1}A_{1,h1}x[0]$ Notice that each $q$-length CSS $1_h 0_h^{q-1}$ in an ACESS (for any $q \in \mathbb{Z}^+$) is defined for a certain sampling period $h$. This essentially represents a closed loop, discretized with sampling period $h$. We percieve this as a *subsystem* $\theta_{q,h}$, for any $q \in \mathbb{Z}^+$. Its augmented system state vector $X$ evolves by computing the feedback control action at $t = kh$, using the controller designed with $h$ sampling period. The controller does not execute in the next $q-1$ sampling periods. The last computed control at $t = kh$ is actuated once in every $h$ duration for the next $qh$ duration. 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. $\theta_{1,h_i} \rightarrow \theta_{3,h_i} \rightarrow \theta_{2,h_j} \rightarrow \theta_{1,h_j}$. When a stable closed-loop, discretized with $h_i$ sampling period and running under certain CSS $1_{h_i}0_{h_i}^{qi'-1}$, starts following another CSS $1_{h_j}0_{h_j}^{qj'-1}$ and another sampling period $h_j$ such that stabilizing LQR and Kalman gains can be designed for the new sampling period, unstable transient behaviour may still exist [11]. To overcome this, we need to constrain the switching to certain sampling periods and certain CSSs corresponding to them and define certain switching rules. In the next section, we discuss how such rules and constraints are synthesized to ascertain the desired stability criteria in such a switched system.

### *C. Stability Analysis of Aperiodic Control Executions as Switched Systems*

Let $\Theta = \{\theta_1, \theta_2, \ldots\}$ be a set of subsystems where any $q$-th 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 $10^{i-1}, i < N$. $\mathcal{H}, N$ 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 $q \in H \times N$, such that all possible combinations from $H \times N$ are present in $\Theta$. We assume each of these subsystems is linear in nature (follows Eq. 3 and 4). We present the detailed stability analysis for a switched system comprising subsystems in $\Theta$ in a discrete-time setting. We start representing such a system as a switched linear system like below.

$$X[k+1] = A_{\sigma[k]}(X[k]),\ \sigma : k \mapsto \mathbb{N},\ k \geq 0, \tag{5}$$

Here $\sigma$ is a switching signal. $\sigma(k) = q$ signifies that at $k$-th sampling period the system is in $q$-th subsystem $\theta_q$, i.e., $X$ evolves using a controller designed with $h$ sampling period 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 $\theta_q$ within $k'$-th and $k$-th sampling period. Thus,

$$N_{\sigma_q}(k', k) \leq N_{0q} + T_q(k', k)/\tau_{d_q} \tag{6}$$

Where $N_{0q}$ is chattering bound for $\theta_q$, $T_q(k', k)$ is the total time spent in $\theta_q$, and $\tau_{d_q}$ is a minimum time duration that the switched system stays in the subsystem $\theta_q$. Notice 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)* [20]. The MDADT for $\theta_q$ is denoted with $\tau_{d_q}$. For constraining the MDADTs of a set of subsystems (eg., $\Theta$) to ensure a desired performance bound while switching within them, we use *Multiple Lyapunov Functions (MLFs)* as explained next.

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 $\kappa_1, \kappa_2$, 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 2 different subsystems, then $\forall \theta_q \in \Theta$ the following holds.

$$\kappa_1(||x(k)||) \leq V_q(x(k)) \leq \kappa_2(||x(k)||) \quad (7)$$

$$\Delta V_q(x(k)) \leq \alpha_q \; V_q(x(k)) \;\; \text{s.t. } \alpha_q \neq 0 \quad (8)$$

$$V_q(x(k)) \leq \mu_q \; V_{q'}(x(k^-)) \; \forall \theta_{q'} \in \Theta \;\; \text{for } \mu_q > 1 \quad (9)$$

In simpler words, $V_q$-s are sub-system wise Multiple Lyapunov Functions (MLFs) [23] respecting Eq. 7-9. In Eq. 8, for stable subsystems $0 < 1 + \alpha_q < 1$ and for unstable subsystems $1+\alpha_q > 0$ [20]. Extending the MDADT-based GUAS stability criteria from Theorem.2 of [20], we claim the following in order to maintain the desired GUES stability during slowly switching between subsystems in $\Theta$.

**Claim 1.** *For the switched system in Eq. 5 having subsystems with MLF $V_q(X[k])$ satisfying Eq. 7-9, the following criteria need to be satisfied in order to ensure a desired GUES margin $\gamma$ while slowly switching between a set of subsystems: (i) for every $q$-th subsystem, there exists a lower bound of the corresponding MDADT $\tau_{d_q} \geq \frac{\ln \mu_q}{|\ln(1+\alpha_q)|}$ and (ii) the switching should follow a minimum* dwell time ratio $v = \frac{\ln \gamma^+ - \ln \gamma}{\ln \gamma - \ln \gamma^-}$ *between the total dwelling duration at stable and unstable subsystems, where $\gamma^- = \max_{q\in\Theta^-}\left[(1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}}\right]$ and $\gamma^+ = \max_{q\in\Theta^+}\left[(1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}}\right]$.*

*Proof.* From Eq. 8, 9, for $k \in [k_p, k_{p+1})$ we can write,

$$V_{\sigma(k)}(x(k)) \leq (1+\alpha_{\sigma(k_p)})^{T_{\sigma(k_p)}(k_p,k)} \mu_{\sigma(k_p)} V_{\sigma(k_p^-)}(x(k_p^-)) \quad (10)$$

Unwinding Eq. 10 over the switching interval $[k_0, k)$ will have the parameters $\mu_q$ and $\alpha_q$ repeated in the above equation as many times as the $q$-th subsystem $\theta_i$ will be switched into. Hence, using the definition of *MDADT* in Eq.6, and the total number of subsystems in $\Theta$ as $\mathcal{M}$, we can re-write the above equation as we get,

$$\begin{aligned} V_{\sigma(k)}(x(k)) &\leq \mu_{\sigma(k)}^{N_{\sigma(p)}(k_p,k)} (1+\alpha_{\sigma(p)})^{T_{\sigma(p)}(k_p,k)} \mu_{\sigma(k_p)}^{N_{\sigma p}(k_{p-1},k_p)} \\ &\quad (1+\alpha_{\sigma(p-1)})^{T_{\sigma(p-1)}(k_{p-1},k_p)} \ldots \mu_{\sigma(1)}^{N_{\sigma(0)}(k_0,k_1)} \\ &\quad (1+\alpha_{\sigma(0)})^{T_{\sigma(0)}(k_0,k_1)} V_{\sigma(k_0)}(x(k_0)) \mu_{\sigma(k)}^{N_{\sigma_q}(k_0,k)} \\ &\leq \prod_{q=1}^{\mathcal{M}} \left(\mu_q^{N_{\sigma_q}(k_0,k)} (1+\alpha_q)^{T_q(k_0,k)}\right) V_{\sigma(k_0)}(x(k_0)) \\ &\leq e^{\left\{\sum_{q=1}^{\mathcal{M}} N_{0q} \ln \mu_q\right\}} \prod_{q\in\Theta^-} \left((1+\alpha_i)\mu_q^{\frac{1}{\tau_{d_q}}}\right)^{T_q(k_0,k)} \\ &\quad \prod_{q\in\Theta^+} \left((1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}}\right)^{T_q(k_0,k)} V_{\sigma(k_0)}(x(k_0)) \end{aligned}$$

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

$$T^- = \sum_{q\in\Theta^-} T_i(k_0,k) \;,\; \gamma^- = \max_{q\in\Theta^-}\left[\ln\left[(1+\alpha_q)\mu_q^{\frac{1}{\tau_{d_q}}}\right]\right] \quad (11)$$

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

$$\text{Hence, } (e^{\gamma^- T^-} \times e^{\gamma^+ T^+}) \leq e^{\gamma(k-k_0)}, \quad (13)$$

$$\begin{aligned} \Rightarrow V_{\sigma(t)}(x(t)) &\leq \mathcal{K} e^{(-\gamma^- T^- + \gamma^+ T^+)} V_{\sigma(k_0)}(x(k_0)) \\ &\leq \mathcal{K} e^{[\gamma(k-k_0)]} V_{\sigma(k_0)}(x(k_0)) \end{aligned} \quad (14)$$

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

$$\mu_q^{\frac{1}{\tau_{d_q}}} \leq \frac{1}{(1+\alpha_q)} \quad \Rightarrow \tau_{d_q} \geq \frac{\ln \mu_q}{|\ln(1+\alpha_q)|} \quad (15)$$

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

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

Note that $T^-$ and $T^+$ represent the total running time into the stable and unstable subsystems, respectively. Therefore, from Eq. 13, we have the *dwell time ratio*, $v$, between the stable and unstable subsystems as, $v = \frac{T^-}{T^+} \geq \frac{\gamma^+ - \gamma}{\gamma - \gamma^-}$. From Eq.14 and the definition of GUES [23], we can conclude that $V_{\sigma(k)}(x(k))$ converges to zero with the desired margin of $\gamma$ as sampling instance $k \to \infty$, and consequently we get the lower bound of the *MDADT* $\tau_{d_q}$ for $q \in \{1, 2, \ldots, \mathcal{M}\}$. □

In case of *fast switching*, we claim the following.

**Claim 2.** *A switched system Eq. 5, that arbitrarily switches between subsystems should have a common Lyapunov function (CLF) for all its switchable sub-systems that satisfy Eq. 7, Eq. 8 and $V_q(x(k)) = V_{q'}(x(k^-)) \; \forall \theta_q, \theta_q' \in \Theta_{clf}$ s.t. $\Theta_{clf} \subseteq \Theta$ in order to maintain a desired GUES decay margin of $\gamma$ while switching among them arbitrarily [23].* □

*Proof.* The proof of this theorem can be established following the previous proof, considering the $\mu_q = 1$ for all $q$-th subsystem in $\Theta_{clf}$, which is a subset of all subsystems $\Theta$ (mandates a *zero* dwell time or arbitrary switching to and from each subsystem$\in \Theta_{clf}$) [11]. □

### D. Safe & Stable Switching Rule Computation for Subsystems

To achieve a desired decay rate $\gamma$ while switching between multiple subsystems $\in \Theta$, we can calculate required *MDADT* $\tau_{d_q}$ using Eq. 15 $\forall q \in \mathcal{H} \times N$ such that $X(t) \in \mathcal{R}_{safe}, \forall t, t_0 \in \mathbb{R}, t \geq t_0$, i.e., the augmented system 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 *LMI*s have a positive definite solution for $P_q$, given the values of $\alpha_q = \lambda^2_{max,q} - 1, \ \forall \theta_q, \theta_{q'} \in \Theta$:

$$A_q^T P_q A_q - P_q \leq \alpha_q P_q, \ P_q \leq \mu_q P_{q'}, \ P_q > 0 \tag{17}$$

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 $P > 0$ solution on solving the LMIs in Eq. 17 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 states within a safe operating region, the switching is also expected to maintain *safety*. In the next section, we proceed to model an extended timed automaton from the designed switched system that can generate such safe and stable ACESS in the form of stable switching sequences.

### E. Formalization of Switched System as Automaton for Safe and Stable ACESS Synthesis

Let, $\mathbf{\Theta} = \Theta_{clf} \cup \Theta_{mlf}$, where $\Theta_{clf}$ is the set of subsystems that have a CLF (supports Claim 2) and $\Theta_{mlf}$ is the set of subsystems that can only have MLFs (does not have a CLF and supports Claim 1) given the GUES decay rate $\gamma$. Note that some subsystem $\theta_q$ can belong to both in $\Theta_{clf}$ and $\Theta_{mlf}$ 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 Sec. III-D (see Eq. 17), we can calculate the *MDADT* $\tau_{d_q}$ for each such subsystem $\theta_q \in \Theta_{mlf}$ (that we denote with $\theta_{q,mlf}$) and the *dwell time ratio* $v$. 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 sampling period for $\theta_q$. We construct an extended timed automaton that can generate all possible sub-system switching sequences respecting the input stability margin and a length requirement. Such traces of this proposed automation are essentially timed sequences of subsystem switching or CSS switching sequences. It maintains the required decay by ensuring *dwell time* and *dwell time ratio*. To ensure this, the automaton imposes proper guards, resets, and location invariants using suitable clocks and variables while switching between CSS. Since such a CSS/subsystem switching sequence generates a safe and stable control execution skipping sequence, we term it Control Skipping Automaton (CSA). In Fig. 3, we provide a schematic structure for this CSA $\mathcal{T}$ realized with $m'$ stabilizable locations in $\Theta_{clf}$ and $m$ locations in $\Theta_{mlf}$. We reuse notations of the subsystems as the notations for the locations in CSA to express their correspondence. The automaton is defined below.

**Definition 4 (Control Skipping Automaton).** *A Control Skipping Automaton (**CSA**) for a control loop is an extended timed automaton $\mathcal{T} = \langle \mathcal{L}, \{\theta_0\}, \{\theta_0\}, \mathcal{C}, V, \mathcal{E}, Inv, \rangle$ where*

- *$\mathcal{L} = \mathbf{\Theta} \cup \{\theta_0\}$ is the finite set of locations that denotes the underlying subsystems.*
- *$\theta_0$ 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 ACESSs with $0$ length and $0$ minimum dwell time and helps us synthesize ACESSs starting from any subsystem.*
- *$\mathcal{C} = \{c, c', p, p'\}$ are the set of clocks used to keep track of global and local times and total dwelling time in stable subsystems, total dwelling time in unstable subsystems, respectively.*
- *$V = \{a, b, s\}$ is a set of variables other than clocks (as it is an extended timed automaton) that are used to keep track of total ACESS length, the count of $0$s, and whether the destination subsystem is stable ($s = 0$) or not, respectively.*
- *$Inv(\theta_q) = \langle Inv^q_{safety}, Inv^q_{len}, Inv^q_{dtr} \rangle$ is the invariant tuple at a q-th location/subsystem. For $Inv^q_{safety}$ and $Inv^q_{len}$, the subscripts refer to the rule it enforces at each $\theta_q \in \mathcal{L}$. $Inv_{dtr}$ enforces a maximum dwell time if $\theta_q$ is unstable such that a desired $ACESS$ length is respected. All of them should hold $true$ to stay in $\theta_q$.*
- *$\mathcal{E} \subseteq \{(\mathcal{L} \setminus \Theta_{mlf}) \times \mathcal{G} \times \mathcal{R} \times (\mathcal{L} \setminus \Theta_{mlf})\} \cup \{(\mathcal{L} \setminus \Theta_{clf}) \times \mathcal{G} \times \mathcal{R} \times (\mathcal{L} \setminus \Theta_{clf})\}$ is the set of transitions/edges. A transition from $\theta_q$ to $\theta'_q$ 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 $\mathcal{E}_{00} \in \mathcal{E}$.*
- *The guard conditions are defined as,*

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

*Here $\mathcal{G}^{qq'}_{len}$ is true when the transition between 2 locations in $\Theta_{mlf} \cup \theta_0$ or between 2 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 $\theta'_q$ cannot be covered or the* dwell time ratio *cannot be maintained (in case of unstable $\theta'_q$), for the desired length of ACCESS $\mathcal{G}^{qq'}_{len}$ 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 $\theta_q$ to $\theta'_q$. Note that the minimum dwell time for a $\theta_{q,clf}$ is $ih$ if $\theta_q = \theta_{i,h}$ (for $\theta_0$ it is $0$). $\mathcal{G}^{qq'}_{\Theta^+}$ is evaluated when the destination location $\theta'_q$ 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 $p, p' \in \mathcal{C}$.*

- *The reset maps are defined as,* $\mathcal{R}_{qq'} = \mathcal{R}_{len}^{qq'} \wedge \mathcal{R}_{dtr}^{qq'} \wedge \mathcal{R}_{s}^{qq'} \wedge \mathcal{R}_{\tau_d}^{qq'}$ *Here,* $\mathcal{R}_{len}^{qq'}$ *is used to update the* $a, b \in V$ *to count the length and number of zeros in the ACESS during the current transition.* $\mathcal{R}_{s}^{qq'}$ *resets the indicator variable* $s \in V$ *if* $\theta_{q'}$ *is stable or* $\theta_0$ *and sets it otherwise. And* $\mathcal{R}_{dtr}^{qq'}$, $\mathcal{R}_{\tau_d}^{qq'}$ *update* $p, p' \in \mathcal{C}$ *and* $c, c' \in \mathcal{C}$ *to maintain current dwell time ratio and dwelling time of* $\theta_q$ *during the current transition, respectively.* □

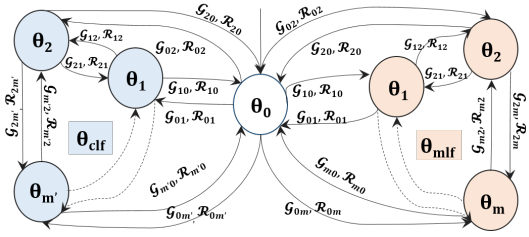



Figure 3: Schematic of CSA

From a language theoretic point of view, we designate $\theta_0$ as the starting and accepting state of the automaton by ensuring the guard $\mathcal{G}_{0q}, \mathcal{G}_{q0}\ \forall \theta_q \in \Theta \setminus \theta_0$. These location invariants and transition guards are generated based on the computations done for a closed loop control task in Sec. III-D. Therefore, *an ACESS satisfying the original GUES requirement* $\gamma$ *within a safe operating region* $\mathcal{R}_{safe}$ 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 $\theta_0$.

**Example 2.** *As an example, for a given length of* 8*, consider a cyclic trace of CSA designed to maintain a GUES of* 85% *is* $(\theta_0, a, b, c, c', p, p', s = 0) \rightarrow (\theta_{1,mlf}, c = 0.1, c' = 0.1, a = 5, b = 0, p = 0.1, p' = 0, s = 1) \rightarrow (\theta_{4,mlf}, c = 0.16, c' = 0.06, a = 8, b = 2, p = 0.1, p' = 0.06, s = 0) \rightarrow (\theta_0, c = 0.16, c' = 0, a = 8, b = 2, p = 0.1, p' = 0.06, s = 0)$*. Here* $\{\theta_0, \theta_{1,mlf}, \theta_{2,mlf}, \theta_{3,mlf}, \theta_{4,mlf}, \theta_{1,clf}, \theta_{2,clf}, \theta_{3,clf}\} \in \mathcal{L}$*, with* $\theta_1 = \theta_{1,0.02}$*,* $\theta_2 = \theta_{2,0.02}$*,* $\theta_3 = \theta_{2,0.06}$ *and* $\theta_4 = \theta_{3,0.02}$*. Among these, the first three are stable, and the last one is an unstable subsystem. The MDADTs for* $\theta_1, \theta_2, \theta_3$ *be* $0.06, 0.04, 0.12$ *sec, respectively, while the* dwell time ratio *is* $1.5$ *to maintain a GUES of* 85%*. Note that, as desired, an* 8 *length ACESS generated from the aforementioned cyclic trace of the CSA is '*$(1_{0.02})^5 1_{0.02}(0_{0.02})^2$*'. Starting from* $\theta_0$*, this sequence spends* $c' = 0.1$ *sec in* $\theta_1$ *which evaluates to* $(1_{0.2})^5$*. Following this, it enters an unstable subsystem* $\Theta_4$*. Since* $(i)$ *as per the* dwell time ratio *constraint* $\mathcal{G}^{14}_{\Theta^+}$*, it is allowed to remain there for* $\lceil \frac{value(p)/1.5}{h=0.02} \rceil \times 0.02$ *sec,* $(ii)$ *and as per* $\mathcal{G}^{14}_{len}$ *this does not violate the desired length, i.e.,* $value(a) + 3 \leq 8$*(remember* $\theta_4 = \theta_{3,0.02}$ *which evaluates to CSS* $1_{0.02}0^2_{0.02}$ *of length* 3*);* $(iii)$ *as per the dwell time constraint* $\mathcal{G}^{14}_{\tau_d}$*,* $value(c') > \tau_{d_1} = 0.6$ *and* $(iv)$ *as per* $\mathcal{G}^{14}_{safety}$*,* $Inv^{4}_{safety}$*,* $x_1(t) \in [-10, 10] \wedge x_2(t) \in [-25, 25]\ \forall t \in [value(c), 0.07)$*, where* $X = [x_1, x_2]^T$ *(i.e. the augmented system states are within a safe operating region in that time window or* $X[k + \lfloor \frac{0.07}{0.02} \rfloor] = A_{100,0.02} X[k],\ X[k+i] \in \mathcal{R}_{safe}, i = 0, 1, 2, 3$*) this is considered to be a valid transition and contributes* $1_{0.02}(0_{0.02})^2$ *to the ACESS. Following this CSA transits to* $\theta_0$ *as the* $value(a) == 8$ *(i.e.,* $\mathcal{G}^{40} = true$*) and we get the ACESS from this times trace* $(1_{0.02})^5 1_{0.02}(0_{0.02})^2$*. Since the whole trace has a fixed sampling period, we can also express this as* 11111100*.*

*Another cyclic timed trace of this CSA can be* $(\theta_0, a, b, c, c', p, p', s = 0) \rightarrow (\theta_{1,clf}, c, c' = 0.08, a = 4, b = 0, p = 0.08, p' = 0, s = 0) \rightarrow (\theta_{3,clf}, c = 0.2, c' = 0.12, a = 8, b = 2, p = 0.12, p' = 0, s = 0) \rightarrow (\theta_0, c = 0.2, c' = 0, a = 8, b = 0, p = 0.2, p' = 0, s = 0)$*, which is generated by transitions within subsystems with CLF, i.e. between* $\{\theta_{1,clf}, \theta_{2,clf}, \theta_{3,clf}\} \mathcal{L} \setminus \Theta_{MLF}$*. The* 8*-length ACESS synthesized from this trace is* $(1_{0.02})^4 (1_{0.06} 0_{0.06})^2$*.*

In the following section, we present an algorithmic representation that can generate ACESSs 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 *ACESSs* for a control task given *certain performance criteria and length requirement*. In real-world CPS, multiple such control programs share a single execution platform along with other tasks. Say, a set of $\mathcal{K}$ control tasks $TS = T_1, T_2, \cdots, T_{\mathcal{K}}$ are to be scheduled dynamically in a shared platform. We choose sets of periodicities $\mathcal{H}^{(j)}$ for each control task $T_j$ such that we can design stabilizable LQR controllers and Kalman filters for the discrete-time closed loop systems with sampling periods $\forall h_i^{(j)} \in \mathcal{H}^{(j)}$. In each case, it must be ensured that the augmented states of these closed loops (i.e., actual and estimated states) remain within a safe operating region $\mathcal{R}_{safe}^{(j)}$. For each of the sampling period $h_i^{(j)} \in \mathcal{H}^{(j)}$, we find the maximum bound of CSS length $N_i^{(j)}$ for which the safety property is maintained. Say the set of maximum CSS bounds for a control task is stored in $N^{(j)} = \{N_1^{(j)}, N_2^{(j)}, \cdots\}$. A periodicity and CSS-length pair $\langle h_i^{(j)}, n_i^{(j)} \rangle$ where $h_i^{(j)} \in \mathcal{H}^{(j)}$ and $n_i^{(j)} \leq N_i^{(j)}$ corresponds to a subsystem or location of the CSA $\mathcal{T}^{(j)}$ for task $T_j$. Given these switchable, safe, and stable subsystems, we now devise an algorithmic framework that helps us assign ACESSs synthesized from the CSA of each control task based on their performance degradation such that utilisation never exceeds a budget. Now, an algorithm designed to look for a valid trace of a finite state automaton takes pseudo-polynomial time to execute, i.e., runtime depends on the number of participating subsystems. However, the guards and invariants in CSA help us prune the invalid traces, which reduces the runtime of such an algorithm. But, for running on an embedded platform used in a CPS, this still might hamper the real-time executions of other tasks which would defeat its primary purpose. Therefore, we rely on some offline preprocessing using the CSAs and curate sets of synthesized ACESSs for each task so that the online algorithm can performance-wise deploy suitable ACESSs in polynomial time.

*1) Offline Preprocessing with Practical Assumptions:* Usually, in CPSs, multiple computation units are connected via the same communication medium to interact with the plant. In such a setup, a worst-case response time analysis helps the designers derive the maximum possible delay in the normal operation for each of these control loops. We refine the sets

of periodicity choices and their corresponding CSS-length bounds for each task accordingly, such that the performance and safety are never compromised even at a critical instance. We denote the refined set of periodicities for task $T_j$ as $\tilde{\mathcal{H}}^{(j)}(\subset \mathcal{H}^{(j)})$ and the refined upper bound of CSS-length for a periodicity $h_i^{(j)} \in \tilde{\mathcal{H}}^{(j)}$ as $\tilde{N}_i^{(j)} < N_i^{(j)}$ ($N_i^{(j)} \in N^{(j)}$) and their set as $\tilde{N}^{(j)}$.

For each of the aforementioned periodicity combinations, there exist different possible hyper-periods (i.e., $lcm$s of the schedulable task periodicities), some of which combinations are not even schedulable under periodic control execution. If we switch between two subsystems with different periodicities following the allowable transitions in CSA, the hyper-period changes. This invalidates the aforementioned static response time analysis. So, *we restrict the transitions of a CSA within the subsystems with the same periodicity choice*. This helps us generate ACESSs for each task with a fixed sampling period to schedule it in a hyper-period. This, in turn, also helps us decide the length of the ACESSs that are to be synthesized from the CSAs, i.e. desired length $l$= the number of execution instances of a control task in a hyperperiod.

When we need to switch between different ACESSs in consecutive hyper-periods, we need to know which of the subsystems are arbitrarily switchable without compromising the safety and performance criteria. Since we are synthesizing ACESSs for the next hyperperiod, we can change the periodicity as well. Therefore, for each of the aforementioned refined set of subsystems, we find the set of arbitrarily switchable subsystems (with the same or different periodicity) and store in a hash map where the key is the current subsystem $\theta_{(h_i^{(j)},n_i^{(j)})}$ and the value contains a set of subsystems $\Theta_{(h_i^{(j)},n_i^{(j)}),clf}$. Any subsystem in $\Theta_{(h_i^{(j)},n_i^{(j)}),clf}$ is arbitrarily switchable from $\theta_{(h_i^{(j)},n_i^{(j)})}$. We denote the list containing the sets $\Theta_{(h_i^{(j)},n_i^{(j)}),clf}$ for all $\theta_{(h_i^{(j)},n_i^{(j)})}$ as $\Theta_{start}^{(j)}$.

$$\Theta_{start}^{(j)}=\{\langle\theta_{(h_i^{(j)},n_i^{(j)})} : \Theta_{(h_i^{(j)},n_i^{(j)}),clf}\rangle \cdots \forall n_i^{(j)}<\tilde{N}_i^{(j)}, h_i^{(j)}\in\tilde{\mathcal{H}}^{(j)}\}$$

We define a method GETNEXTLOC($\Theta_{start}^{(j)}, \rho_j$) that outputs the set of subsystems from the list $\Theta_{start}^{(j)}$, that can be arbitrarily switched into, starting from the subsystem that was visited last in an ACESS $\rho_j$.

Since we decided to keep the periodicity choice of a task constant through a hyper-period, the length of ACESS remains fixed for that hyper-period and period choice. We can work out all such possible hyper-periods and their corresponding ACESS-lengths corresponding to different periodicity choices of each task in $TS$. The number of possible hyper-period choices depends on the number of schedulable periodicity choices of the other tasks in $TS$. Let us consider there is $M$ number of such hyperperiod choices for $TS$. There can exist different ACESS length choices for $T_j \in TS$ given a fixed hyperperiod choice, corresponding to the sampling period choices of $T_j$. Then, we can generate all possible ACESSs for each of these length-choices from the CSA $\mathcal{T}^{(j)}$. We compute their processor utilisations and group them w.r.t their processor utilisations. Each group is then sorted in ascending order of utilisation. We further group these ACESSs w.r.t. their starting subsystem (after $\theta_0$). Each group with the same utilisation and starting subsystem is again subgrouped w.r.t their hyper-periods and sorted by the ascending order of hyper-period-duration. This way, we prepare maps of maps $\mathcal{M}^{(j)}$ with the tuple containing utilisation ($util$) and starting subsystem ($\theta_{start}$, after $\theta_0$) pair as the key and another list of maps as values. The inner map uses hyper-periods ($HP$) as key, and the the ACESS length (#len) and the corresponding set of ACESSs with this specification as values like below.

$$\mathcal{M}^{(j)}=\{\langle\ util\downarrow_1,\ \theta_{start}\rangle :\{HP\downarrow_2:\{\langle\#len\downarrow_3,\{\rho_1,\rho_2\cdots\}\rangle,\cdots\},\cdots\},\cdots\}$$

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 with $\downarrow_3$. Using this data structure for each task $T_j \in TS$, we can synthesize all possible ACESSs from its CSA for a given $(i)$ utilisation bound, $(ii)$ a starting subsystem, $(iii)$ a desired length and $(iv)$ a bound on the number of 0s.

We create method GETACESSMIN($\mathcal{M}^{(j)}, util_{min}, \theta_{start}, \#len_{min}$) to generate the list of all possible ACESSs with minimum utilisation $util_{min}$, starting from the subsystem $\theta_{start}$ and having a minimum length of $\#len_{min}$, grouped by hyper-periods in the linear order from $\mathcal{M}^{(j)}$. We create another method GETACESSMAX($\mathcal{M}^{(j)}, util_{max}, \theta_{start}, \#len_{max}$) to generate the list of all possible ACESSs with maximum utilisation $util_{max}$, starting from the subsystem $\theta_{start}$ and having a maximum length $\#len_{max}$ grouped by hyper-periods in the linear order of the length of the inner list of maps in $\mathcal{M}^{(j)}$. Now, we present an online algorithm to dynamically schedule ACESSs for each task based on their performance degradation using these data structures.

*2) Performance-aware Dynamic Scheduling:* For a feedback-driven ACESS generation from CSA of each of these control tasks, based on their performance degradation and utilisation budget, we propose Algo. 1. Performance degradation caused by intentional or unintentional external disturbances increases the overall control cost. In such scenarios, the controller needs to be executed at a higher frequency (i.e. more 1's in the corresponding $ACESSs$) to reduce the performance degradation and, thereby, the control cost. Consequently, it demands higher processor utilisation. Algo. 1 first computes the utilisation demands according to the current control cost of each task in descending order of the task priority. Doing so, if the total utilisation budget is surpassed, Algo. 1 adjusts utilisation allocation starting from the lower priority tasks. Subsequently, it synthesizes the $ACESSs$ of the control tasks according to the new utilisation demands. Algo. 1 is demonstrated in details below.

The algorithm takes the following inputs. **(i)** the set of $\mathcal{K}$ control tasks $TS = T_1, T_2, \cdots, T_{\mathcal{K}}$ that are to be scheduled dynamically in a given platform. **(ii)** a list of ACESSs $\mathcal{M}^{(j)}$ for each $T_j \in TS$ sorted in the order of utilisation, length, etc., **(iii)** the list of arbitrarily switchable subsystems $\Theta_{start}^{(j)}$ 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.

**(iv)** Current task specifications $Specs = \{spec^{(j)} | spec^{(j)} = \langle h^{(j)}, \rho^{(j)}, c^{(j)}, J^{(j)}, J^{(j)}_{ub}, J^{(j)}_{lb} \rangle$. Here $h^{(j)}, \rho^{(j)}, c^{(j)}$ are the current periodicity, the current ACESS used, and the execution time for a task $T_j$. $J^{(j)}, J^{(j)}_{ub}, J^{(j)}_{lb}$ are the current LQR cost, the upper bound and lower bounds for the LQR cost for $T_j$, and **(v)** utilisation budget $U_b$ depending on the scheduling policy.

**Input** Task set $TS = T_1, T_2, \cdots, T_{\mathcal{K}}$ to be co-scheduled, $\mathcal{M}^{(j)}$, $\Theta^{(j)}_{start}$ for each $T_j \in TS\}$, list of current task specifications $Specs = \{spec^{(j)} | spec^{(j)} = \langle h^{(j)}, \rho^{(j)}, c^{(j)}, J^{(j)}, J^{(j)}_{ub}, J^{(j)}_{lb} \rangle \forall T_j \in TS\}$, utilisation budget $U_b$
**Output** An updated $Specs$ with schedulable ACESSs $\{\langle h^{(j)}, \rho^{(j)} \rangle \forall T_j \in TS\}$
1: $utils, newUtils \leftarrow$ empty array, *commonHPSet*$\leftarrow$ empty set ▷ Initialization
2: $sort(TS, priority, descending)$
3: **for** each $j < \mathcal{K}$ **do** ▷ New util demand calculation
4: $utils[j] \leftarrow \frac{c^{(j)} \times \#one(\rho^{(j)})}{HP}$ ▷ compute task wise Utils.
5: $J^{(j)} \leftarrow$ Update $J$ for current state following Eq. 2 ▷ compute LQR costs
6: **if** $J^{(j)} \geq J^{(j)}_{ub}$ **then** ▷ if High LQR Cost
7: $newUtils[j] \leftarrow utils[j] \times J^{(j)}/J^{(j)}_{ub}$ ▷ util. increment
8: **else** $newUtils[j] \leftarrow utils[j]$ ▷ Assign the last util
9: **if** SUM$(newUtils) \geq U_b$ **then** ▷ If utilisation beyond budget
10: **for** each $j < \mathcal{K}$ **do** ▷ Util. reduction to meet the total budget
11: **if** $J^{(\mathcal{K}-j)} \leq J^{(\mathcal{K}-j)}_{lb}$ **then** ▷ if Low LQR Cost
12: $newUtils[\mathcal{K}-j] \leftarrow utils[j] \times J^{(\mathcal{K}-j)}/J^{(\mathcal{K}-j)}_{lb}$ ▷ util. reduction
13: **if** SUM$(newUtils) \geq U_b$ **then** ▷ If utilisation still beyond budget
14: $utilAdjFact \leftarrow U_b$/SUM$(newUtils)$
15: $newUtils \leftarrow utilAdjFact \times newUtils$ ▷ scale down Util.
16: **for** each $T_i \in TS$ **do**
17: $\Theta_0[j] \leftarrow$ GETNEXTLOC$(\Theta^{(j)}_{start}, \rho^{(j)})$ ▷ next switchable subsystems
18: **if** $newUtils[j] > utils[j]$ **then**
19: $\mathcal{M}[j] \leftarrow$ GETACESSMIN$(\mathcal{M}^{(j)}, newUtils[j], \Theta_0[\mathcal{K}-j], \#len(\rho^{(j)}))$
20: **else**
21: $\mathcal{M}[j] \leftarrow$ GETACESSMAX$(\mathcal{M}^{(j)}, newUtils[j], \Theta_0[\mathcal{K}-j], \#len(\rho^{(j)}))$
22: *commonHPSet* $\leftarrow$ *commonHPSet* $\cap \mathcal{M}[j].key$ ▷ find common HP
23: $HP \leftarrow CommonHPSet[0]$ ▷ choose next common HP for all tasks
24: **for** each $T_i \in TS$ **do**
25: $h^{(j)}, \rho^{(j)} \leftarrow \mathcal{M}[j][HP][0]$ ▷ deploy ACESSs with new HP, new utils.
26: **return** $Specs$

Algorithm 1: Performance-aware Dynamic Task Scheduling

We start by initializing new arrays $utils, newUtils$ to store the current and desired utilisation of each task and an empty set $commonHPSet$ to store the set of hyperperiods that the tasks agree upon (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 Eq. 2 and store in $J^{(j)}$ (line 5). We compute the utilisation of the $j$-th task depending on the ACESS it is running with, i.e., $\rho^{(j)}$ and its WCET $c^{(j)}$ and store it in $utils$ (see line 4). If the control cost of a higher priority task is beyond its tolerable upper bound (see line 6), we need to assign higher utilisation to them. For this, we increase the task-wise utilisation by the ratio between the actual cost and the cost upper bound and store them in $newUtils$ (see line 7). Since the control cost of this task is beyond the tolerable upper bound of the cost, the multiplying factor is always $> 1$ which ensures the newly assigned utilisation is higher than the current utilisation (both are naturally less than $U_b$). If doing so 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 9-12). If the LQR cost for a lower priority task is below its cost lower bound, the utilisation of the task is reduced by a factor of {(its control cost)/(the cost lower bound)} (see line 12). If the total of the newly assigned task utilisations is still more than the utilisation budget $U_b$ (line 13), utilisation for each task is scaled down by a factor of $U_b$/total of newly assigned utilisation for each task in $TS$ (see line 15).

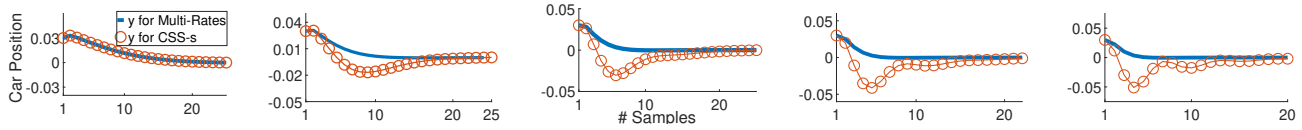
In lines 16-22, we derive possible sets of ACESSs for each task based on the redistributed utilisation based on the control degradation. First, for each task, we fetch the list of subsystems from which we can start in the next hyper-period. We use the GETNEXTLOC() method to compute the arbitrarily switchable subsystems from the last subsystem visited by $\rho^{(j)}$ and store them in $\Theta_0[j]$ (line 17). We fetch the ACESSs for the tasks that are assigned with increased utilisation using the GETACESSMIN() method. As mentioned earlier, given the set of curated ACESSs for the $j$-th control task (synthesized from its CSA) in the form of a list of maps $\mathcal{M}^{(j)}$, the GETACESSMIN() method gives us a set of ACESSs having minimum utilisation $newUtils[j]$, starting from $\Theta_{start}[j]$ and with a minimum length as $\rho^{(j)}$ (see line 19, all of these are provided as inputs). They are stored in $\mathcal{M}[j]$ as a map, having the hyper-periods as keys and sorted in their ascending order. We store the ACESSs for the tasks that are assigned with reduced utilisation using the GETACESSMAX() method and store it in $\mathcal{M}[j]$ in similar format (line 21).

Note that, based on the newly computed ACESSs, each task can have multiple possible hyper-period choices from the set $M$ of all possible hyper-periods where the length of the hyper-period must be the multiple of the ACESSs' lengths. Now, we need to decide the next hyper-period in such a way that it is multiple of the ACESS length of all control tasks. Therefore, we store the common set of hyper-periods for all tasks in $commonHPSet$ by finding the intersections of the sets of keys in $\mathcal{M}[j] \forall T_j \in TS$ (see line 22). In lines 24-25, we find an ACESS from $\mathcal{M}[j]$ stored against this common hyper-period key. We update the specification of each task $specs^{(j)}$ with this ACESS and its corresponding sampling period in line 25 (remember, we use ACESSs having a fixed sampling period in a single hyper-period) and finally return the updated task specifications $Specs$ (in line 26). Each task is executed following the newly assigned ACESS in the next hyper period.

**Timing Analysis:** Consider the maximum length among the possible $M$ hyperperiods is $h_{max}$. To compute the number of 1's in an ACESS, it takes $O(h_{max})$ time. Therefore, the line 3-8 takes $O(\mathcal{K}h_{max})$ time. Line 10-12 takes $O(\mathcal{K})$ time. Let, the maximum number of sub-systems among all the control tasks in $n_{max}$. The function GETNEXTLOC (line 17) therefore runs in $O(n_{max})$ time. Similarly, consider the maximum number of entries among all $\mathcal{M}[j]$'s is $m_{max}$. Therefore, both GETACESSMIN() (line 19) and GETACESSMAX() (line 19) take $O(m_{max})$ time. Since the maps $\mathcal{M}[j]$'s are sorted, the intersection to compute $commonHPSet$ runs in $O(m_{max})$ time as well. Therefore, the overall runtime of Algo. 1 is $O(\mathcal{K}(h_{max} + n_{max} + m_{max}))$.

## IV. EXPERIMENTAL RESULTS

In this section, we demonstrate the experimental results for in-depth analysis of the proposed method. We use an automotive test case for this purpose. The safe operation of automotive systems relies on the timely execution of different sets of control tasks running in multiple electronic control




(a) h = 0.02 & CSS =1 (b) h = 0.04 & CSS =10 (c) h = 0.08 & CSS=100 (d) h = 0.1 & CSS=1000 (e) h = 0.12 & CSS=10000

Figure 4: CSSs and Periodicity Changes for Suspension Control

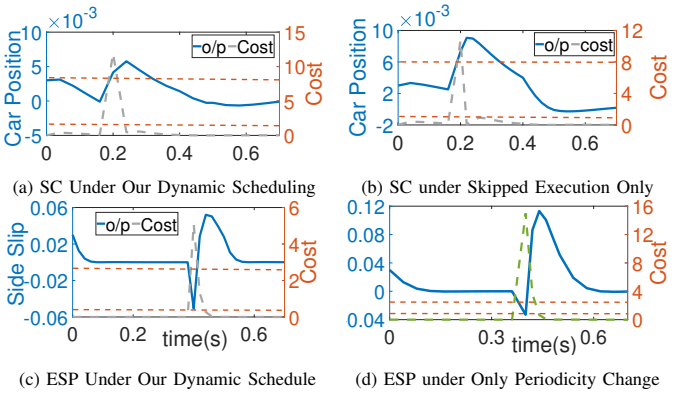
units (ECUs) connected via different network protocols like controller area network (CAN).

**Experimental Setup:** In our experimental setup, we implement 4 such automotive control tasks: electronic stability program (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) [9], [24] (refer to Tab. I col. 1). We implement these control tasks in ARM-based 32-bit Infineon Tricore Aurix-397 ECU, along with several other tasks. We build a CAN setup that connects this ECU with an ETAS Labcar Real-time PC, where we run the vehicle plant models. Following the AUTOSAR mandates [1], we implement separate reception tasks that filter and receive sensor IDs transmitted by the Labcar RTPC in CAN. On 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. We also schedule our proposed algorithm once in every hyper-period at the same core as the other control tasks, which assigns proper activation patterns to the control tasks based on their corresponding performances.

**CSA Synthesis:** We start by deriving the switching parameters for each of the control loops as discussed in Sec. III-D in order to synthesize their CSAs for a desired GUES stability criteria as provided in col. 5 of Tab. 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 $\mathcal{R}_{safe}$ as mentioned in col. 5. We use YALMIP with the Mosek optimization engine to solve the LMIs given in Eq. 17 in order to find out the switchable subset of subsystems with MLFs or CLFs. As a result of these computations for every control loop, in columns 2 and 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 ACESSs as mentioned in Sec. III-F1. In Fig. 4, we demonstrate the behaviour of such safe subsystem choices for suspension control. Each subfigure in Fig. 4 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 periodic control execution (periodicities mentioned in sub-captions), and the red circled plots present the system characteristics under skipped executions (CSSs mentioned in sub-captions). Notice that as we increase the control actuation/execution frequency (from 20ms in the leftmost to 120ms in the rightmost), the system stabilises faster. On the other hand, as we increase the number of consecutively skipped executions after the actuation/control execution at 20ms, the system shows some undershoot but remains within the safe region, i.e. $[-0.1, 0.1]$ (see Tab. I, col.5, row.4). However, notice that among these controllers with multiple sampling periodicity, only the controllers with 20ms, 40ms and 60ms 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.

| Sys. | Subsystems in CLF ⟨h(ms) : CSSs⟩ | Subsystems in MLF ⟨ h (ms):CSSs ⟩ | 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) |

Table I: Implementation Details & Parameters Synthesized for Control Tasks




(a) SC Under Our Dynamic Scheduling (b) SC under Skipped Execution Only

(c) ESP Under Our Dynamic Schedule (d) ESP under Only Periodicity Change

Figure 5: Comparison between Proposed Method & SOTA

**Dynamic Scheduling and Comparison with SOTA:** After the CSA synthesis, we go through the offline pre-processing steps as mentioned in Sec. III-F1. This helps us access the ACESSs generated from each CSA in the linear order of time (i.e. $O(\mathcal{K}(h_{max}+n_{max}+m_{max}))$, see Sec. III-F2) during the real-time operation of Algo. 1. Algo. 1 is implemented in the same core of the control unit where we execute the control

tasks. It executes once in every hyper-period to observe the LQR control cost (performance metric, see Sec. II-1) and accordingly assigns execution schedules or ACESSs with proportional (with cost change) utilisation change. We demonstrate this for the control tasks in our setup, with an example scenario in Fig. 5. In all four subfigures in Fig. 5, we are plotting *system outputs in* $blue$ (car position for SC and side slip for ESP) on the left y-axis and *LQR control costs with dashed grey line* on the right y-axis, w.r.t time (in seconds) in the x-axis. In Fig. 5a and Fig. 5c (left subfigures), we demonstrate system outputs under the proposed dynamic scheduling and in Fig. 5b and Fig. 5d (right subfigures) we demonstrate the effect of SOTA, i.e., only multi-rate scheduling [11], [12] and only control execution skip-based scheduling [9], [14] approaches.

At the start of the scenario, SC is running with the controller for a $40ms$ 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 a tolerable upper bound of $8$ (marked with orange dashed lines in top subfigures, Fig. 5a and Fig. 5b). In this situation, the online Algo. 1 assigns more $(4\%)$ utilisation by deploying the $20ms$ 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 ACESSs $(1_{0.04})^5$ (i.e., with the same sampling period of 40ms). As can be seen, by using our approach, the output settles faster (Notice the settling of the blue plot in Fig. 5a compared to Fig. 5b). The ESP task, on the other hand, runs with a controller of 20ms sampling period and with ACESS $1010101010$ (utilisation $2.5\%$) till $0.02ms$ 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 5c and Fig. 5d). Hence, to provide more utilisation to the SC, Algo. 1 assigns ACESS $1000110010$ (utilisation $2\%$) to ESP and continues the same till 0.04s. Whereas under the SOTA strategies that solely rely on multi-rate 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 Tab. I col.2, row.1). Due to sudden arrival/discovery of an obstacle at 0.04 seconds, there is a sudden deviation in side slip angle causing a high control cost for ESP beyond the tolerable upper bound of $2.5$ (Notice the orange dashed lines in bottom subfigures). Our methodology deploys ACESS $(1_{0.02})^{12}$ (i.e. $111111111111$ ACESS under 20ms sampling period, with $5\%$ utilisation), and the SOTA multi-rate strategy switches to $20ms$ controller (as $40$ms and $20$ms controllers have CLF respecting given GUES, see Tab. I) assigning the same $5\%$ utilisation. But as can be seen in Fig. 5d, the periodicity switching causes a larger overshoot, causing $doubled$ control cost compared to our strategy (Fig. 5c). Under similar scenarios with noise/disturbances, how our strategy works better compared to the SOTA for the other control tasks is shown in [25]. This helps us successfully demonstrate there can be such situations where utilising the multi-rate and control skips together might result in a better performance-aware dynamic scheduling of control tasks in an embedded control unit.

## V. Conclusion

We provide a framework for sub-system identification and finitary representation of switching constraints by considering under a common umbrella the existing paradigms of multi-rate as well as weakly hard scheduling of control tasks. This is then leveraged to efficiently co-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) other use cases from industrial benchmarks, and 2) more complex multi-core platform mappings.

## References


[1] S. Kramer *et al.*, "Real world automotive benchmarks for free," in *WATERS*, vol. 130, 2015.

[2] K. . Arzen *et al.*, "An introduction to control and scheduling co-design," in *CDC*, vol. 5, 2000.

[3] D. Henriksson *et al.*, "Optimal on-line sampling period assignment for real-time control tasks based on plant state information," in *CDC*, IEEE, 2005.

[4] E. Bini *et al.*, "Delay-aware period assignment in control systems," in *RTSS*, IEEE, 2008.

[5] A. Cervin *et al.*, "Optimal online sampling period assignment: Theory and experiments," *IEEE transactions on control systems technology*, vol. 19, no. 4, pp. 902–910, 2010.

[6] G. Bernat *et al.*, "Weakly hard real-time systems," *IEEE transactions on Computers*, 2001.

[7] M. Hamdaoui *et al.*, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE transactions on Computers*, vol. 44, 1995.

[8] M. S. Branicky *et al.*, "Scheduling and feedback co-design for networked control systems," in *CDC*, vol. 2, pp. 1211–1217, IEEE, 2002.

[9] S. Ghosh *et al.*, "A structured methodology for pattern based adaptive scheduling in embedded control," *ACM TECS*, 2017.

[10] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE Transactions on parallel and distributed systems*, 1999.

[11] M. Schinkel *et al.*, "Optimal control for systems with varying sampling rate," in *ACC*, IEEE, 2002.

[12] P. Marti *et al.*, "Optimal state feedback based resource allocation for resource-constrained control tasks," in *RTSS*, IEEE, 2004.

[13] R. Castané *et al.*, "Resource management for control tasks based on the transient dynamics of closed-loop systems," in *ECRTS*, IEEE, 2006.

[14] M. Maggio *et al.*, "Control-system stability under consecutive deadline misses constraints," in *ECRTS*, 2020.

[15] S. Linsenmayer and F. Allgower, "Stabilization of networked control systems with weakly hard real-time dropout description," in *CDC*, 2017.

[16] M. B. Gaid, A. Cela, Y. Hamam, and C. Ionete, "Optimal scheduling of control tasks with state feedback resource allocation," in *2006 American Control Conference*, pp. 6–pp, IEEE, 2006.

[17] H. Liang *et al.*, "Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees," in *ICCAD*, 2020.

[18] N. Vreman *et al.*, "Stability of linear systems under extended weakly-hard constraints," *IEEE Control Systems Letters*, vol. 6, 2022.

[19] D. Soudbakhsh *et al.*, "Co-design of control and platform with dropped signals," in *ICCPS*, ACM, 2013.

[20] X. Zhao *et al.*, "Stability and stabilization of switched linear systems with mode-dependent average dwell time," *IEEE TAC*, 2012.

[21] G. Zhai *et al.*, "Stability analysis of switched systems with stable and unstable subsystems: an average dwell time approach," *International Journal of Systems Science*, 2001.

[22] K. J. Åström and B. Wittenmark, *Computer-controlled systems*. Prentice-Hall, Inc., 1997.

[23] D. Liberzon, *Switching in systems and control*. Springer, 2003.

[24] S. Adhikary *et al.*, "Skip to secure: Securing cyber-physical control loops with intentionally skipped executions," in *Joint Workshop on CPS&IoT Security and Privacy*, 2020.

[25] "Anonymous Git Repo." https://anonymous.4open.science/r/DynamicSchedulingCSA-8213/README.md, 2024.