Timing Debugging for Cyber-Physical Systems

Debayan Roy*, Clara Hobbs†, James H. Anderson†, Marco Caccamo*, Samarjit Chakraborty†
*Technical University of Munich, Germany, †University of North Carolina at Chapel Hill, USA
Email: *{debayan.roy, mcaccamo}@tum.de, †{cghobbs,anderson,samarjit}@cs.unc.edu

***Abstract*—This paper is concerned with the following question: *Given a set of control tasks that are not schedulable, i.e., their required timing properties cannot be satisfied, what should be changed?* While the real-time systems literature proposes many different schedulability analysis techniques, it surprisingly provides almost no guidelines on what should be *changed* to make a task set schedulable, when it is not. We show that when the tasks in question are control tasks, this *timing debugging* question in the context of cyber-physical systems (CPS) may be answered by exploiting the dynamics of the physical systems that these control tasks are expected to influence. Towards this, we study a very simple setup, *viz.*, when a set of periodic tasks with implicit deadlines is not schedulable, by how much should the periods be changed in order to make the task set schedulable? Among the many ways in which the periods can be modified, our proposed strategy is to change the periods in a manner such that while the task set becomes schedulable, the *poles* of the closed-loop system experience the minimal shift. Since the poles influence the closed loop dynamics of the system, we thereby ensure that we obtain a system with the desired timing properties whose dynamics is very similar to the dynamics of the original (non-schedulable) system. We formulate this CPS timing debugging strategy as an optimization problem and illustrate it with a concrete example.**

I. Introduction

Validation of timing properties is a crucial step in the design of many safety-critical systems. For this, the real-time systems literature provides a wide range of *schedulability analysis* techniques [1]. In the same measure, there are also many different choices of *scheduling algorithms* to ensure that the desired timing properties are satisfied. However, the third important pillar in the design of time-critical systems, *viz.*, *"what to change when timing properties are not satisfied?"* is surprisingly very weak in terms of the relevant techniques and literature that is available. The reason for this being there can be *many* different changes that could satisfy timing constraints, but their feasibility or impact on the functional properties of the system being designed cannot be evaluated using the abstractions typically used for timing validation. The goal of this paper is to address this important question of *"what should be changed?"* and to propose a new class of *timing debugging* techniques. They rely on the observation that many time-critical software tasks implement feedback control loops, and therefore the changes to satisfy schedulability constraints could be evaluated from a *system-level* perspective, *viz.*, how do those changes impact the *dynamics* of the feedback control loop? Given this integrated approach of ensuring the timing properties of software tasks by accounting for the (physical) dynamics of the system being controlled, one can view this as a cyber-physical systems (CPS) approach to timing debugging.

The problem we address in this paper is the most basic timing debugging problem: Given a set of periodic tasks with implicit deadlines, *i.e.*, deadlines equal to periods, which periods should be changed in order to make the task set schedulable? See Fig. 1. Since the earliest deadline first (EDF) scheduling policy is known to be optimal in this setting, we consider this for scheduling the tasks on a single processor. Our goal in this paper is merely to initiate a discussion on this approach, and towards this we show that even this simplest setup opens up many different design issues. A more real-life setup could involve multiple processors and communication buses on which different tasks are mapped and scheduled. The timing debugging question could then expand to: how should task and message mapping and scheduling, periods of these tasks, the granularity with which they are partitioned, and possibly even their offsets, be changed in order to satisfy the required timing and schedulability constraints?

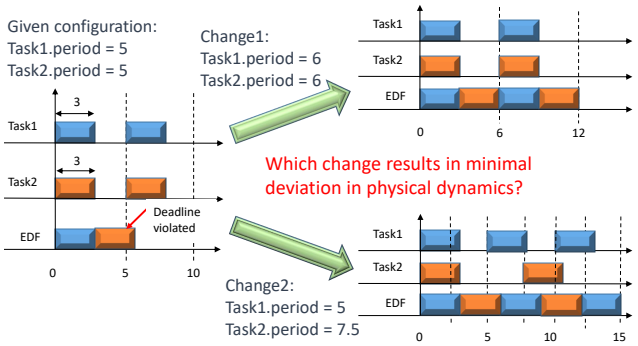

Fig. 1. The simplest timing debugging problem.

Our proposed approach in this paper hinges on the fact that the dynamics of the closed-loop system depend on the location of the system *poles* [2], and by changing the task periods these poles also shift. Therefore, a first approach could be to change those task periods that result in the minimum shift in the poles, which translates into an optimization problem. However, a system has multiple poles and depending on their location they impact the system dynamics differently. A change in the task period might have different impact on how much each of these poles change. Similarly, how should the notion of *dynamics* be quantified? They can be quantified using metrics like peak overshoot, settling time, or the root sum square difference between the trajectories of how the state of the system evolves for two different periods. How these metrics depend on the shift in the system poles is complex and system dependent. All of these aspects lead to the complexity of the debugging process, which we aim to highlight in this paper. In addition to system dynamics, the *input saturation* constraints or the energy requirements associated with the controller might also be impacted with changing timing behavior, thereby adding another dimension to this problem. Given the exploratory nature of this paper, we restrict our study to the case, where we do *not* redesign the controllers when their timing properties change.

The design flow assumed in this paper is the usual one—first the control strategy, along with its associated parameters (*e.g.*, sampling periods and gain values) are designed. This is followed by their implementation as software tasks, and in this process the schedulability test fails. The goal now is to adjust their periods in order to make them schedulable, while minimally changing their targeted dynamics (as planned by the control designers in the first step). Changing the task periods *together* with the gain values leads to a *co-design* problem that has been studied recently in different setups [3]–[5].

Contributions and related work: While different forms of scheduling control tasks have been studied in the past [6], [7], the problem of *timing debugging of CPS*, to the best of our knowledge, has not been investigated before in spite of its fundamental importance. Our goal is to initiate this study with system dynamics providing the guidance for timing debugging decisions [8]. Using a simple setup, we aim to illustrate the different facets of this problem and lay the groundwork for investigating for involved setups.

The problem of determining *timing contracts* that satisfy control performance, followed by synthesizing a schedule to satisfy those contracts have been studied in [9]. Similarly, [10] dealt with time-triggered scheduling of control tasks. But these works are orthogonal to the problem we study here, where the controllers and their schedules are given, but they are not feasible. *Online* period adaptation is considered in [11] where historical operational data are used to predict how changes in the task period will influence control performance. Such a technique could also be integrated within our scheme for timing debugging where offline simulations could be used for performance prediction. When model checking is used for timing validation, counterexamples generated in the case of timing violations can serve as cues for debugging [12]. Such techniques can be combined with the approach we propose here. Many schedulability tests—such as the utilization bound test considered here—do not return any counterexamples, and in such cases, our approach might be the only guidance available for property refinement to meet schedulability constraints.

Organization: In the next section, we introduce the mathematical models of feedback control systems and processor scheduling. In Sec. III, using an example, we demonstrate the complex relation between physical dynamics and system poles. In Sec. IV, we discuss a few preliminary metrics based on the closed-loop poles that can guide the timing debugging process, and we, further, propose an optimization algorithm to search for the task periods that will minimize the deviation in the physical dynamics. In Sec. V, we apply different heuristics on a case study and show that the location of the closed-loop poles can guide the timing debugging process if appropriately used. Sec. VI provides the concluding remarks.

II. Background

We study a CPS setting comprising a set of control tasks running on a single processing unit. Each task implements a feedback controller for a physical plant. In this section, we introduce the mathematical models concerning processor scheduling and feedback control systems.

Schedulability of control tasks: We consider that the processor runs the tasks based on EDF scheduling policy. A task τ_i is represented by a tuple $\{e_i, p_i\}$, where e_i is the worst-case execution time (WCET) and p_i is the period. Here, we assume implicit-deadline tasks, i.e., the deadline is equal to the period. Let $\mathbb{T}$ represent the set of tasks running on the processor. The schedulability of the tasks in $\mathbb{T}$ can be evaluated using a utilization bound test as follows:

$$U = \sum_{\tau_i \in \mathbb{T}} \frac{e_i}{p_i} \leq 1. \tag{1}$$

Here, U is the processor utilization.

Plant model: In this work, we study linear and time-invariant (LTI) physical plants for which the continuous-time mathematical model can be written as follows:

$$\dot{x}(t) = Ax(t) + Bu(t); \quad y(t) = Cx(t). \tag{2}$$

Here, $x(t)$, $u(t)$, and $y(t)$ represent system states, control input, and plant output, respectively, at time t. A, B, and C are constant system matrices.

For an embedded implementation of the controller, the control input is periodically calculated based on the sampled state of the plant. The software task that calculates the control input takes a non-negligible time for computation, especially in a resource-constrained platform. Moreover, we assume EDF scheduling on the processor, and therefore, a control task may have to wait for or be preempted by higher-priority tasks. Thus, there is a delay from sampling to the time when the control input is ready. This delay is upper-bounded by the task's deadline in a schedulable system. Considering that the controller is implemented following the logical execution time (LET) paradigm [13], [14], the control input is applied to the plant at the task's deadline. Thus, we get a fixed sampling-to-actuation delay that enables precise control predictions.

For a control task τ_i, the sampling period h_i is equal to the task period p_i, i.e., $h_i = p_i$. For a zero-order hold implementation, the sampled-data model of the plant for one-sample delay can be written as follows [15]:

$$\begin{aligned} &x[k+1] = \Phi x[k] + \Gamma u[k-1]; \quad y[k] = Cx[k]; \\ &\text{where } \Phi = e^{Ah}, \quad \Gamma = \int_0^h e^{As} ds \cdot B. \end{aligned} \tag{3}$$

Here, k is a sampling instant.

For an augmented state vector given by $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$, Eq. (3) can be reformulated as follows:

$$\begin{aligned} &z[k+1] = \Phi_a z[k] + \Gamma_a u[k]; \quad y[k] = C_a z[k], \\ &\text{where, } \Phi_a = \begin{bmatrix} \Phi & \Gamma \\ \mathbf{0} & \mathbf{0} \end{bmatrix}; \; \Gamma_a = \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix}; \; C_a = \begin{bmatrix} C & 0 \end{bmatrix}. \end{aligned} \tag{4}$$

Here, $\mathbf{0}$ and $\mathbf{I}$ are zero matrix and identity matrix respectively.

Feedback control law: We study stabilization control where the control input is calculated based on the feedback control law that is given by

$$u[k] = -Kz[k]. \tag{5}$$

Here, K is the feedback control gain. In this work, the control law that a task implements is known a priori. Note that the feedback control law can be obtained using standard techniques, e.g., pole placement (Chapter 5 in [16]) and linear-quadratic regulator (LQR) (Chapter 11 in [16]). From Eq. (4) and (5),

we derive the closed-loop system model as follows:

$$z[k+1] = (\Phi_a - \Gamma_a K)z[k] = \Phi_{cl} z[k]. \quad (6)$$

Here, Φ_{cl} is the closed-loop state-transition matrix. The eigenvalues of Φ_{cl} are the poles of the closed-loop system. For an asymptotically stable system, each pole $\rho_j \in \mathbb{P}$ must lie inside a unit circle, i.e., $|\rho_j| < 1$. These poles also influence the system response. The farther the pole is from the origin in the z-plane, the slower is the system response. For second or higher order systems with multiple closed-loop poles, the pole that is farthest to the origin is called the *dominant pole*, which mostly determines the speed with which the system stabilizes.

Performance measures: Typically, the control response is characterized by rise time, settling time, overshoot, and input energy. For stabilization control, let us assume that the goal is to maintain the system output y at 0. We further consider that a disturbance brings the output to y_d. Here we follow [16], where it is stated in page 370, "the effect of a disturbance on a linear system can be analyzed as an initial-value problem." We are interested in the system response after a disturbance. Thus, we define the performance measures accordingly. (i) Rise time (T_r) is the time taken the system response to go from $90\,\%$ of y_d to $10\,\%$ of y_d. (ii) Settling time (T_s) is the time taken by the system response to reach and stay within $[-0.01y_d, 0.01y_d]$. (iii) Assuming y_d is positive, overshoot (O) is defined based on the minimum value y_{min} attained by y, i.e., $O = \frac{|y_{min}|}{y_d} \cdot 100\,\%$. (iv) Input energy E_u is defined based on the control inputs applied to the plant to reject the disturbance, i.e., $E_u = h \cdot \sum_{k=0}^{\infty} |u[k]|$.

III. A Motivational Example

We study a second-order DC motor speed control system [17] for which the continuous-time plant model is given as follows:

$$\dot{x}(t) = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u(t); \;\; y(t) = [1 \quad 0]\,(t). \quad (7)$$

For a sampling period $h = 20\,\text{ms}$ and one-sample delay, we obtain two different control laws as follows:

$$\mathcal{K}_S: \quad u[k] = -\begin{bmatrix} 0.0084 & 0.7448 & -0.6205 \end{bmatrix} z[k],$$
$$\mathcal{K}_F: \quad u[k] = -\begin{bmatrix} 216.8020 & 24.6962 & 0.8795 \end{bmatrix} z[k].$$

$\mathcal{K}_S$ and $\mathcal{K}_F$ place the three closed-loop poles at 0.8 and 0.3, respectively. As shown in Fig. 2(a), $\mathcal{K}_S$ leads to a slower response compared to $\mathcal{K}_F$ as the poles are farther from the origin.

We increment the sampling period h in a step of 1 ms from $h = 20\,\text{ms}$ until $h = 200\,\text{ms}$. For each increment, we note the new closed-loop poles for both $\mathcal{K}_S$ and $\mathcal{K}_F$ respectively. Let us denote the new poles as $\{\rho_1', \rho_2', \rho_3'\}$, where $|\rho_1'| \geq |\rho_2'| \geq |\rho_3'|$, and the expected poles (i.e., for $h = 20$ ms) as $\{\rho_1, \rho_2, \rho_3\}$, where $|\rho_1| \geq |\rho_2| \geq |\rho_3|$. We calculate the displacement D of the poles as $D = |\rho_1 - \rho_1'| + |\rho_2 - \rho_2'| + |\rho_3 - \rho_3'|$. We further calculate the displacement D^* of the dominant pole as $D^* = \big||\rho_1| - |\rho_1'|\big|$. Here, D^* measures the change in the distance of the dominant pole from the origin.

For both $\mathcal{K}_S$ and $\mathcal{K}_F$, we plot D and D^* with change in the sampling period in Fig. 2(b). It can be observed that for the slow controller $\mathcal{K}_S$, changing the sampling period does not displace the poles as much as in case of the fast controller $\mathcal{K}_F$. Moreover, using $\mathcal{K}_F$, the closed-loop system becomes unstable

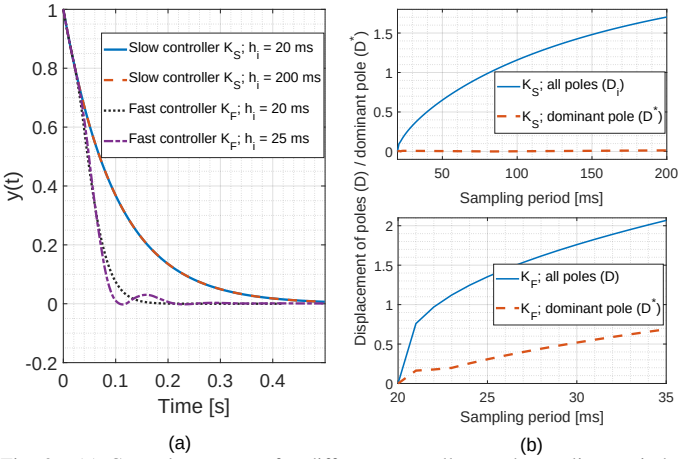

Fig. 2. (a) Control responses for different controllers and sampling periods. (b) Displacement vs sampling period for different control laws.

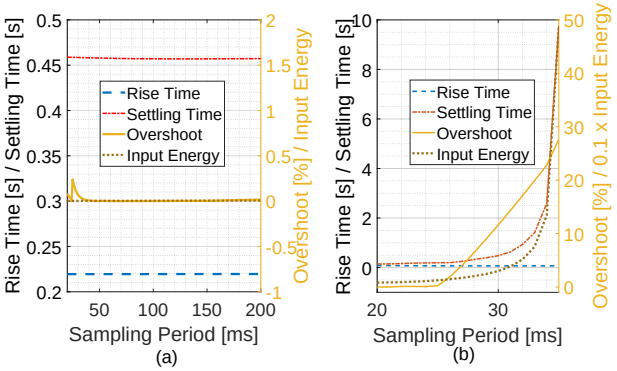

Fig. 3. Rise time, settling time, overshoot, and input energy vs sampling period for (a) a slow controller $\mathcal{K}_S$ and (b) a fast controller $\mathcal{K}_F$.

if the sampling period is increased beyond 35 ms. Further, note that using $\mathcal{K}_S$, the distance of the dominant pole from the origin changes negligibly with the increase in the sampling period.

We also simulate the plant with both $\mathcal{K}_S$ and $\mathcal{K}_F$ respectively while varying the sampling period, and we note down the rise time, settling time, overshoot, and input energy. Fig. 3 shows the variation of these measures with sampling period for both $\mathcal{K}_S$ and $\mathcal{K}_F$. It can be observed that the deviation is insignificant for $\mathcal{K}_S$ even if the sampling period is increased to 200 ms, i.e., 10 times the value of the sampling period $h = 20\,\text{ms}$ for which the control law is originally synthesized. This might be because D^* remains close to 0 even at such high sampling period values. While the rise time remains fairly constant for K_F, the settling time, overshoot, and input energy deviate significantly with increase in the sampling period for $\mathcal{K}_F$.

To further visualize the deviation in the physical dynamics, we plot, in Fig. 2(a), the responses when (i) $\mathcal{K}_S$ is used at $h = 200\,\text{ms}$ and (ii) $\mathcal{K}_F$ is used at $h = 25\,\text{ms}$. We denote Δ_y as the root sum square difference in plant response. That is, $\Delta_y(\mathcal{K}, h, h') = \sqrt{\sum_0^\infty (y[k'] - y'[k'])^2}$, where $y[k']$ and $y'[k']$ are the responses when $\mathcal{K}$ is used at sampling period h and h' respectively. Here, k' denotes a discrete time instant, and for our experiments, we assume that two consecutive instants are separated by 0.1 ms. For the responses in Fig. 2(a), we get $\Delta_y(\mathcal{K}_S, 20, 200) = 0.022$ and $\Delta_y(\mathcal{K}_F, 20, 25) = 1.412$. This is also reflected in the figure where there is no noticeable deviation in the responses corresponding to $\mathcal{K}_S$ while there is a significant deviation in the responses corresponding to $\mathcal{K}_F$.

In summary, we have three main takeaways from this example: (i) Each control loop might have a different sensitivity to the sampling period. During timing debugging, it is important to increase the periods of the control tasks that are less sensitive to the change. (ii) Both displacement of the poles and deviation in the physical dynamics might increase with increasing sampling period, however, the rate of change can be very different. Thus, the relation between the change in physical dynamics and the shift in the closed-loop poles can be challenging to model. (iii) While a certain characteristic of system response might remain unchanged, others might deviate appreciably with a change in the sampling period. This multi-dimensional aspect makes the timing debugging even more challenging.

IV. Heuristics for Timing Debugging

We consider a set of tasks $\mathbb{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$ that is mapped on a processing resource. Each task τ_i implements a controller $\mathcal{K}_i$ that controls a physical plant (A_i, B_i, C_i). $\mathcal{K}_i$ is obtained for a sampling period h_i and one sample delay. Thus, τ_i is expected to run with a period $p_i = h_i$, and accordingly, the closed-loop poles will be placed at $\{\rho_{i,1}, \rho_{i,2}, \ldots, \rho_{i,\nu_i}\}$, where $|\rho_{i,1}| \geq |\rho_{i,2}| \geq \cdots \geq |\rho_{i,\nu_i}|$. We assume that the WCET e_i of a task $\tau_i \in \mathbb{T}$ is known a priori.

Problem formulation for timing debugging: In this work, we look into the following question: *What should be changed if the task set $\mathbb{T}$ is not schedulable?* In our problem setting, $\mathbb{T}$ is not schedulable if Eq. (1) is violated, i.e., the processor utilization U is greater than 1. WCETs of the tasks cannot be changed as the processor architecture is fixed. Thus, U can be reduced only by increasing the task periods. Therefore the problem boils down to determining the period p'_i with which each task $\tau_i \in \mathbb{T}$ should run such that $\mathbb{T}$ becomes schedulable.

There are numerous possible ways in which the periods of the tasks can be changed such that Eq. (1) holds. However, the goal is to find the option for which the dynamics of the physical plants, that these tasks control, change minimally. Let $\mathcal{F}$ represent a function that captures the change in the physical dynamics of the controlled plants. The timing debugging then essentially becomes a constrained optimization problem where the task periods $\{p'_1, p'_2, \cdots, p'_n\}$ are the variables, Eq. (1) is a constraint, and the objective is to minimize $\mathcal{F}$.

Now, the main challenge in timing debugging is to formulate $\mathcal{F}$ appropriately. From control theory, we know that plant dynamics depend on the closed-loop poles. Hence, an intuitive approach would be to change the task periods in such a way that the closed-loop poles experience minimal shift. We denote that the closed-loop poles shift to $\{\rho'_{i,1}, \rho'_{i,2}, \cdots, \rho'_{i,\nu_i}\}$ for a task τ_i when the period is changed to p'_i. We assume that $|\rho'_{i,1}| \geq |\rho'_{i,2}| \geq \cdots \geq |\rho'_{i,\nu_i}|$. In this work, we evaluate three simple formulations for $\mathcal{F}$.

As studied in Sec. III, the first formulation considers the change in the magnitude of the dominant pole. We denote this change as D_i^* for a task τ_i. Thus, $\mathcal{F}$ is given by:

$$\mathcal{F} = \sum_{\tau_i \in \mathbb{T}} D_i^*, \text{ where : } D_i^* = \Big| |\rho_{i,1}| - |\rho'_{i,1}| \Big|. \tag{8}$$

Next, we consider the displacement between the set of new poles $\{\rho'_{i,1}, \rho'_{i,2}, \cdots, \rho'_{i,\nu_i}\}$ and the corresponding expected poles $\{\rho_{i,1}, \rho_{i,2}, \ldots, \rho_{i,\nu_i}\}$. We denote this displacement as D_i for a task τ_i. Accordingly, we formulate $\mathcal{F}$ as follows:

$$\mathcal{F} = \sum_{\tau_i \in \mathbb{T}} D_i, \text{ where : } D_i = \frac{\sum_{j=1}^{\nu_i} |\rho'_{i,j} - \rho_{i,j}|}{\nu_i}. \tag{9}$$

Furthermore, we take into consideration how the control response and the dominant pole change with the sampling period. Let $\widehat{D}_i$ capture the change in the physical dynamics of the plant corresponding to τ_i. Now, we formulate $\mathcal{F}$ as follows:

$$\mathcal{F} = \sum_{\tau_i \in \mathbb{T}} \widehat{D}_i, \text{ where : } \widehat{D}_i = (D_i^* \cdot |\rho_{i,1}|)^2. \tag{10}$$

We have observed that when the dominant pole $\rho_{i,1}$ is farther away from the origin, the dynamics changes faster with respect to D_i^*. Thus, we multiply D_i^* with $|\rho_{i,1}|$ (i.e, the distance of the dominant pole from the origin) to normalize this rate of change based on the position of the dominant pole. Furthermore, in Fig. 2(b), it can be seen that the rate of increase in D_i^* reduces with the increase in the sampling period in case of $\mathcal{K}_\mathcal{F}$. On the other hand, the change in the dynamics (e.g., settling time and overshoot) becomes more rapid at higher values of sampling period, as seen in Fig. 3(b). To appropriately represent the relation between the change in the dynamics and D_i^*, we raise the term $(D_i^* \cdot |\rho_{i,1}|)$ to the power 2, as shown in Eq. (10).

Optimization algorithm: The relation between the closed-loop poles and the period of a task is not linear. Also, the expression for $\mathcal{F}$ is non-linear in the task periods. Thus, it is challenging to search for the set of task periods such that $\mathcal{F}$ is minimized.

We propose a dynamic programming approach in Algorithm 1 to search for the optimal set of task periods $\mathcal{P}$ that ensure schedulability. We assume that task periods can be

Algorithm 1: Optimal timing debugging

```
Input  : {{A_i, B_i, C_i, K_i, e_i, p_i} | ∀ τ_i ∈ T}
Output: P
Ω[1].P = {p_1, p_2, ··· , p_n};
Ω[1].U = Σ_{i=1}^{n} e_i / Ω[1].P[i];
Ω[1].F = 0;
while ∃_i Ω[i].U > 1 do
    Ω' = {};
    for i ← 1 to |Ω| do
        if Ω[i].U ≤ 1 then
            Ω'.add( Ω[i] );
        else
            for j ← 1 to n do
                Ω* = Ω[i];
                Ω*.P[j] = Ω*.P[j] + Δ_p;
                if stable(Ω*.P[j]) == true then
                    Ω*.U = Σ_{k=1}^{n} e_k / Ω*.P[k];
                    Ω*.F = calculateF( Ω*.P );
                    Ω'.add( Ω* );
                end
            end
        end
    end
    Ω = {};
    for i ← 1 to |Ω'| do
        if (∄_j Ω'[j].U ≤ Ω'[i].U ∧ Ω'[j].F ≤ Ω'[i].F ∧ Ω'[j] ≠
          Ω'[i]) then
            Ω.add( Ω'[i] );
        end
    end
end
i* = arg min_{1≤j≤|Ω|} Ω[j].F;
return P = Ω[i*].P;
```

TABLE I
CASE STUDY SPECIFICATION

τ_i	$\{A_i, B_i, C_i\}$	$\{\rho_{i,j}\}$	$\mathcal{K}_i$
τ_1	Eq.(7)	$\begin{bmatrix}0.5\\0.5\\0.5\end{bmatrix}$	$\begin{bmatrix}122.8023\\16.7514\\0.3311\end{bmatrix}^T$
τ_2	$A=\begin{bmatrix}-0.2 & 0.67\\-10 & -100\end{bmatrix}$, $B=\begin{bmatrix}0\\37000\end{bmatrix}$, $C=\begin{bmatrix}1 & 0\end{bmatrix}$	$\begin{bmatrix}0.4\\0.4\\0.4\end{bmatrix}$	$\begin{bmatrix}0.1365\\0.0009\\0.1655\end{bmatrix}^T$
τ_3	$A=\begin{bmatrix}-10 & 1\\-0.2 & -15\end{bmatrix}$, $B=\begin{bmatrix}0\\20\end{bmatrix}$, $C=\begin{bmatrix}1 & 0\end{bmatrix}$	$\begin{bmatrix}0.7\\0.7\\0.7\end{bmatrix}$	$\begin{bmatrix}0.2938\\0.0566\\-0.5405\end{bmatrix}^T$
τ_4	$A=\begin{bmatrix}0 & 1 & 0\\0 & -1.0865 & 8.4872\cdot10^3\\0 & -9.9636\cdot10^3 & -1.4545\cdot10^6\end{bmatrix}$, $B=\begin{bmatrix}0 & 0 & 3.6364\cdot10^5\end{bmatrix}^T$, $C=\begin{bmatrix}1 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}0.7\\0.7\\0.7\\0.7\end{bmatrix}$	$\begin{bmatrix}0.0091\\0.0201\\5.6765\\-1.6308\end{bmatrix}^T$
τ_5	$A=\begin{bmatrix}0 & 1 & 0\\0 & -0.0227 & 54.5455\\0 & -35.2857 & -70\end{bmatrix}$, $B=\begin{bmatrix}0 & 0 & 28.1754\end{bmatrix}^T$, $C=\begin{bmatrix}1 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}0.3\\0.3\\0.3\\0.3\end{bmatrix}$	$\begin{bmatrix}23.3252\\0.8360\\0.6791\\0.4576\end{bmatrix}^T$

τ_i	e_i [ms]	p_i [ms]	T_r [s]	T_s [s]	O [%]	E_u
τ_1	5	15	0.1	0.18	0	5.363
τ_2	3	10	0.05	0.1	0	0.004
τ_3	6	20	0.2	0.38	0	0.056
τ_4	8	30	0.41	0.84	0	0.028
τ_5	7	25	0.1	0.23	0	1.253

incremented only in discrete steps of size Δ_p. We start with the expected task periods $\{p_1, p_2, \cdots, p_n\}$ and determine the processor utilization corresponding to them (lines 1–2). $\mathcal{F}$ is 0 for this initial case (line 3). Note that Ω can hold sets of task periods together with the utilization U and the objective value F. We iteratively expand Ω by incrementing the period of one task at a time by Δ_p; this continues until the processor utilization corresponding to each set in Ω is at most 1 (lines 4–27). In each iteration, Ω' is initialized as a null set (line 5). We go through the sets in Ω one by one (lines 6–20). If the utilization corresponding to a set $\Omega[i]$ is acceptable, then we do not need to change the periods any more and add the set directly to Ω' (lines 7–8). Otherwise, we increment the task periods in $\Omega[i]$ one by one and add the feasible new combinations to Ω' (lines 9–19). Now, we set Ω to a null set (line 21). In lines 22–26, we insert only those sets from Ω' to Ω that are not dominated by any other set. That is, if for a set of task periods $\Omega'[i].P$, there exists another set $\Omega'[j].P$ that has a lower utilization and a lower objective value, then $\Omega'[i]$ is not added to Ω. At the end, we determine the set that has the lowest objective value which is also the optimal solution (lines 28–29).

V. EXPERIMENTAL RESULTS

We consider a case study where 5 control tasks are mapped on a single processing unit. Each task controls a physical plant. The 5 physical plants come from DC motor speed and position control systems [17]. The continuous-time plant models ($\{A_i, B_i, C_i\}$) are provided in Table I.

The execution times (e_i) of the control tasks are assumed as given in Table I. A feedback control law is synthesized for each task for a given task period using pole placement. The task periods (p_i), the control gains ($\mathcal{K}_i$), and the closed-loop poles ($\{\rho_{i,1}, \rho_{i,2}, \cdots, \rho_{i,\nu_i}\}$) are tabulated in Table I. We simulate each plant for a sampling period $h_i = p_i$ and the obtained control gains $\mathcal{K}_i$. The rise time (T_r), settling time (T_s), overshoot (O), and input energy (E_u) are noted in Table I. When the control tasks are implemented with the expected periods, the utilization is $U = \frac{5}{15} + \frac{3}{10} + \frac{6}{20} + \frac{8}{30} + \frac{7}{25} = 1.48 > 1$. Thus, the given configuration is not feasible.

TABLE II
RESULTS USING NAÏVE APPROACH

τ_i	p_i' [ms]	T_r' [s]	T_s' [s]	O' [%]	E_u'	Δ_y
τ_1	20	0.09	0.21	0	7.182	0.88
τ_2	15	0.03	0.15	9	0.005	2.87
τ_3	30	0.2	0.36	0	0.059	0.15
τ_4	50	0.32	2.52	19.7	0.05	14.77
τ_5	40	0.07	0.36	14.1	1.702	5.69

TABLE III
RESULTS COMPARISON FOR DIFFERENT APPROACHES

	Naïve	$\mathcal{F}(D_i^*)/\mathcal{F}(D_i)$	$\mathcal{F}(\widehat{D}_i)$
(p_i') ms	Table II	$(15, 15, 65, 85, 25)$	Table IV
$\Delta\overline{T}_r\%$	20.39	10.44	16
$\Delta\overline{T}_s\%$	65.69	2040.25	18.57
$\Delta\overline{O}\%$	8.56	14.68	3.14
$\Delta\overline{E}_u\%$	35.74	676.29	21.47
$\overline{\Delta}_y$	4.872	30.9	1.574

A naïve approach: For this case study, we first consider a naïve approach to changing the period of each task where all tasks contribute almost equally to the reduction in the processor utilization. The solution obtained using this approach is outlined in Table II, along with the rise time, settling time, overshoot, and input energy measured corresponding to the updated period of each task. For τ_4, the deviation in the physical dynamics is the greatest: the settling time becomes almost three times the expected value and there is an overshoot of $19\,\%$. The root sum square difference in the plant response is $\Delta_y = 14.77$, which is significant given that the maximum value of y is 1. The average change in the response characteristics obtained using this naïve approach is provided in Table III.

Proposed heuristics: We also use the metrics discussed in Sec. IV for timing debugging. For each of three objective functions (using D_i^*, D_i, and $\widehat{D}_i$ respectively), we apply Algorithm 1 to find how the task periods should be changed.

When we use D_i^* and D_i respectively to capture the deviation in the physical dynamics, we obtain the same set of task periods as given in Table III. Here, the periods of the tasks τ_3 and τ_4 are predominantly changed, i.e., from $20\,\mathrm{ms}$ to $65\,\mathrm{ms}$ and from $30\,\mathrm{ms}$ to $85\,\mathrm{ms}$ respectively. Note that the original closed-loop poles corresponding to these two tasks are farther away from the origin compared to other tasks, as shown in Table I. As mentioned in Sec. IV, when the original poles are farther from the origin, the physical dynamics change at a faster rate with respect to D_i and D_i^*. As the values of D_i and D_i^* are smaller, it is expected that the dynamics will not change significantly. However, this is not the case with τ_4, where the settling time increases from $0.84\,\mathrm{s}$ to $86\,\mathrm{s}$, the input energy increases from 0.028 to 0.96, and the overshoot becomes $64\,\%$. Due to the large deviation in the plant dynamics for τ_4, the average changes in settling time, overshoot, and input energy are exorbitantly high as given in Table III. The main purpose of showing these results in the paper is to bring into attention that the most intuitive heuristics for timing debugging might not give acceptable results. The main reason for this is the complex relation between the physical dynamics and the closed-loop poles that is challenging to model.

When we use $\widehat{D}_i$ to capture the change in the physical dynamics, we get a solution that is summarized in Table IV.

TABLE IV
RESULTS USING PROPOSED APPROACH $\mathcal{F}(\widehat{D}_i)$

τ_i	p'_i [ms]	T'_r [s]	T'_s [s]	O' [%]	E'_u	Δ_y
τ_1	20	0.09	0.21	0	7.182	0.88
τ_2	15	0.03	0.15	9	0.005	2.87
τ_3	75	0.2	0.33	0.9	0.075	0.73
τ_4	30	0.41	0.84	0	0.028	0
τ_5	35	0.07	0.26	5.8	1.435	3.39

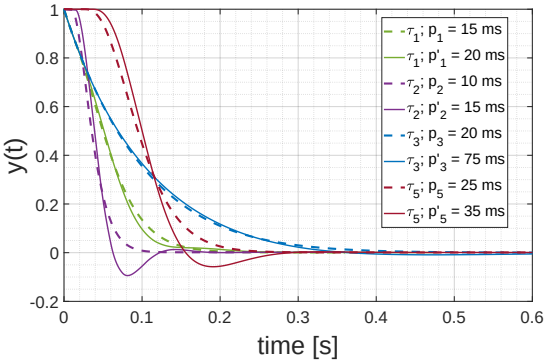

Fig. 4. Change in control responses using the proposed heuristics.

This heuristic successfully determines the task τ_3 for which the dynamics does not change appreciable with increase in the task period. Thus, it changes the period of τ_3 from $20\,\text{ms}$ to $75\,\text{ms}$. The average changes in the rise time, settling time, overshoot, and input energy respectively are provided in Table III. Note that it performs better than the naïve approach in all respects. In particular, the average change in the settling time is reduced from $65.69\,\%$ to $18.57\,\%$ and the average change in the input energy is reduced from $35.74\,\%$ to $21.47\,\%$. These are significant gains. In Fig. 4, control responses corresponding to the expected task periods and the updated periods are shown for better illustration of the change in the dynamics. Control response for τ_4 is not shown as the period is not changed.

Comparison against optimal solution: We perform an exhaustive search to determine the periods for which the average root sum square $\overline{\Delta}_y$ of the difference in the control response is the minimum. We get an optimal solution where the task periods are 25, 15, 115, 30, and 25 respectively (in order) and $\overline{\Delta}_y = 1.265$. Using our proposed heuristic, we get $\overline{\Delta}_y = 1.574$, i.e., $24.43\,\%$ higher. However, for this solution, the average change in the settling time is $37.54\,\%$, i.e., $102.15\,\%$ higher than the solution obtained using our proposed heuristic. On the other hand, when we perform an exhaustive search to minimize the change in the settling time, we get the same results as obtained using our proposed heuristic. This shows that timing debugging for CPS is a multi-dimensional optimization problem.

Performance of Algorithm 1 For all three metrics (i.e., D_i^*, D_i, and $\widehat{D}_i$ respectively), Algorithm 1 gives results within $3\,\text{min}$. The exhaustive searches we performed to find the optimal set of task periods took almost $2\,\text{h}$ to run. Thus, Algorithm 1 reduces the search runtime by approximately $97.5\,\%$. On the other hand, for each of the three metrics, Algorithm 1 gives the optimal solution, i.e., it does not compromise on the optimality to reduce the runtime.

VI. Concluding Remarks

This paper calls attention to the fundamental yet unexplored problem of timing debugging for cyber-physical systems (CPS). Towards solving this problem, we show that the main challenge is to quantify how the dynamics of a physical plant will deviate when the timing properties of the control software is changed. Here, we explore the possibility of exploiting the shift in the closed-loop system poles to capture the deviation in the physical dynamics. Although simpler models involving system poles fail to show promise, a more detailed metric guides appropriately in the debugging process.

Although this work gives some preliminary results, the robustness of the proposed heuristic is yet to be evaluated. Furthermore, we believe that machine learning algorithms can be applied to learn a more complex model that can guide the timing debugger. Given a robust guide, it would be interesting to formulate the timing debugging problem for complex distributed systems with interdependent tasks and messages.

Acknowledgements

Caccamo is supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. The work of Hobbs, Anderson and Chakraborty is supported by the NSF award #2038960.

References

[1] S. Baruah, "Real-time computing," in *Computing Handbook, Third Edition: Computer Science and Software Engineering*, T. F. Gonzalez, J. Diaz-Herrera, and A. Tucker, Eds. CRC Press, 2014, pp. 58: 1–14.
[2] R. C. Dorf and R. H. Bishop, *Modern Control Systems*, 9th ed. USA: Prentice-Hall, Inc., 2000.
[3] D. Goswami, R. Schneider, and S. Chakraborty, "Co-design of cyber-physical systems via controllers with flexible delay constraints," in *ASP-DAC*, 2011.
[4] D. Roy, L. Zhang, W. Chang, and S. Chakraborty, "Automated synthesis of cyber-physical systems from joint controller/architecture specifications," in *FDL*, 2016.
[5] W. Chang, L. Zhang, D. Roy, and S. Chakraborty, *Control/architecture codesign for cyber-physical systems*, ser. Handbook of Hardware/Software Codesign. Springer Netherlands, 2017.
[6] A. Masrur, S. Drössler, T. Pfeuffer, and S. Chakraborty, "Vm-based real-time services for automotive control applications," in *RTCSA*, 2010.
[7] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *J. Syst. Archit.*, vol. 59, no. 10-D, pp. 1215–1230, 2013.
[8] L. Ju, B. K. Huynh, A. Roychoudhury, and S. Chakraborty, "Performance debugging of Esterel specifications," *Real Time Syst.*, vol. 48, no. 5, pp. 570–600, 2012.
[9] M. A. Khatib, A. Girard, and T. Dang, "Scheduling of embedded controllers under timing contracts," in *HSCC*, 2017.
[10] T. Nghiem, G. J. Pappas, R. Alur, and A. Girard, "Time-triggered implementations of dynamic controllers," *ACM TECS*, vol. 11, no. S2, pp. 58:1–58:24, 2012.
[11] X. Dai and A. Burns, "Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems," *Journal of System Architecture*, vol. 103, p. 101691, 2020.
[12] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *J. ACM*, vol. 50, no. 5, pp. 752–794, 2003.
[13] G. Frehse, A. Hamann, S. Quinton, and M. Woehrle, "Formal Analysis of Timing Effects on Closed-Loop Properties of Control Software," in *RTSS*, 2014.
[14] H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu, "Security-driven codesign with weakly-hard constraints for real-time embedded systems," in *ICCD*, 2019.
[15] D. Roy, W. Chang, S. K. Mitter, and S. Chakraborty, "Tighter dimensioning of heterogeneous multi-resource autonomous CPS with control performance guarantees," in *DAC*, 2019.
[16] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems (3rd ed.)*, ser. Prentice Hall Information and System Sciences. Prentice-Hall Inc., 1997.
[17] D. Roy, S. Ghosh, Q. Zhu, M. Caccamo, and S. Chakraborty, "Goodspread: Criticality-aware static scheduling of CPS with multi-QoS resources," in *RTSS*, 2020.