Ensuring Cyber-Physical System Stability in the Presence of Deadline Misses

Martina Maggio

***Abstract*—Microcontrollers are essential in modern devices like smart bikes, cars, and drones. They must execute tasks timely on limited resources, and their complexity increases the risk of missing deadlines, potentially harming the system and users. This paper summarises some years of research in analysing embedded controllers for computational delays that cause deadline misses.**

Very few tools exist for assessing the properties of cyber-physical systems considering their microcontroller implementations as first-class citizens. This research fills this gap with one specific type of timing anomalies: deadline misses. We examine controllers under various timing violations: bursts of deadline misses, constrained misses, and random overruns. We combine these events with the actual details of the implementation of control systems and align the analysis with real-world use.

I. Introduction

Cyber-Physical Systems (CPSs) are integrated networks of digital and physical components. Modern life would be impossible without power grids, transportation systems, spacecraft, and medical devices. In the digital age, these systems operate in an evermore autonomous fashion, making critical decisions with limited information, without human intervention. Even when designed with the best available tools, CPSs are subject to *failures*. A famous and compelling example comes from the Mars Perseverance rover and in particular Flight 6 of its exploration helicopter Ingenuity. In an article[1] appeared on May 27, 2021, NASA reported an anomaly:

"*Approximately 54 seconds into the flight, a glitch occurred in the pipeline of images being delivered by the navigation camera. This glitch caused a single image to be lost, but more importantly, it resulted in all later navigation images being delivered with inaccurate timestamps. From this point on, each time the navigation algorithm performed a correction based on a navigation image, it was operating on the basis of incorrect information about when the image was taken. The resulting inconsistencies significantly degraded the information used to fly the helicopter, leading to estimates being constantly "corrected" to account for phantom errors. Large oscillations ensued.*"

Even with the image timestamping failing, Ingenuity's flight control system was able to maintain stability, and the helicopter was able to land within about five meters of the intended location. Indeed, as written by NASA,[2] they "*designed Ingenuity to tolerate significant errors without becoming unstable, including errors in timing.*" Without an *a priori* analysis of what happens in case of failures, there is no guarantee that the helicopter would not crash, as the possibility of this glitch occurring was not considered in the system design. After experiencing the anomaly, the helicopter team patched the Ingenuity codebase, fixing the anomaly occurred during Flight 6. Yet, Ingenuity faced a different anomaly during Flight 53.[3] These two anomalies did not cause long-term problems to the helicopter's mission, because the design of Ingenuity is remarkably robust at its core. However, few CPSs have an 80 million dollars budget and can afford reaching such a high level of robustness by design.

The lesson that Ingenuity teaches us is that timing anomalies cannot be completely avoided in complex systems. The presence of prior disasters or harsh operating conditions has driven researchers to focus on guaranteeing the reliable functioning of CPSs in the presence of problems. Obtaining these guarantees requires effort from multiple disciplines: control theory offers tools to manage environmental uncertainties [27], [10], computer science comes into play when assessing and constraining computational failures [3], [7], runtime verification ensures that the system remains safe during operation [9]. Yet, carrying out this process to the extent that one needs for high reliability and targeting correctness in the presence of timing anomalies is possible only at a very high cost, or for simple enough CPSs.

If we want to rely on regular CPSs – that are built lacking high budgets and robustness – we need to know how many and which type of timing anomalies they can withstand. In this paper, I will review the history of our research investigation into a specific type of timing anomalies: deadline misses. While this is a single type of problem that CPSs face, ensuring the reliability of CPSs in the presence of deadline misses is a first step to enhance these systems' robustness. The development of more reliable and robust CPS can significantly reduce the risk of accidents and failures in critical systems like transportation (including autonomous vehicles and public transit systems), healthcare (such as life-support machines and surgical robots), and industrial automation. This research has

[1]https://mars.nasa.gov/technology/helicopter/status/305/surviving-an-in-flight-anomaly-what-happened-on-ingenuitys-sixth-flight/

[2]https://mars.nasa.gov/technology/helicopter/status/305/surviving-an-in-flight-anomaly-what-happened-on-ingenuitys-sixth-flight/

[3]In an article (https://mars.nasa.gov/technology/helicopter/status/495/the-long-wait/) published on November 06, 2023, NASA discusses Flight 53 of the helicopter: "*Ingenuity corrects its spatial orientation estimate by tracking the movement of ground features within these navcam images, but these data must be perfectly time-synchronized with measurements from the inertial guidance system to provide valid corrections. In the case of Flight 53, this synchronization step mysteriously failed in a way that hadn't been observed on any of the prior 52 flights on Mars or during the years of ground testing that preceded them. As of this writing, the precise cause is strongly suspected but has not been conclusively proven.*"

979-8-3503-6334-0/24/$31.00 ©2024 IEEE
DOI 10.1109/RAGE62451.2024.00008

extensive societal, economic, environmental, and technological impacts, offering a vision of a future where strong, dependable CPSs improve our quality of life.

II. Preliminaries

CPSs typically interact with the physical environment around them via sensors and actuators. At the core, they calculate control signals that should be applied to the external environment in order to experience a desired outcome, such as a desired trajectory that the Ingenuity helicopter should autonomously follow based on sensor data that reveal its current position estimate. To design these systems, the physical part and the computational part are usually separated by means of interfaces. In the following, we review the models used for plant and controller in the physical and computational sense. This allows us to clearly state the problem that we are trying to solve with our research.

A. Models of Plant and Controller

We describe here the physical model of the system and of its controller. These mathematical models characterise the system's behaviour in time, typically via discrete-time linear time-invariant systems written in *state-space* form, the formalism used in the rest of this paper. The subscript k indicates the k-th iteration of the system evolution (both for the controller and for the plant).

The equation that characterises the plant evolution is

$$\mathcal{P} \; : \; \begin{cases} x_{k+1} = A\,x_k + B\,u_k \\ \quad y_k = C\,x_k + D\,u_k, \end{cases} \tag{1}$$

where u_k, x_k, and y_k are respectively the plant input, the plant state, and the plant output at time k. It is important to note that this is a linear model, that approximates the behaviour of the physical systems. Non-linear effects can typically be neglected around the equilibrium point [2], [21], yet they come into play when the system state is far from the point it is linearised about. Hence, one of the goals of a control system is to keep the physical plant near the equilibrium point, ensuring the validity of the linear model as an approximation of the physical phenomenon that should be controlled.

Controllers are typically defined using the same formalism, with a general linear control law being[4]

$$\mathcal{C} \; : \; \begin{cases} z_{k+1} = F\,z_k + G\,y_k \\ u_{k+1} = H\,z_k + K\,y_k, \end{cases} \tag{2}$$

where z_k is the controller (digital) state. A controller $\mathcal{C}$ is *stateless* (or *static*) if it has no dependence on the internal state z. On the contrary, in a *stateful* (or *dynamic*) controller $\mathcal{C}$, the calculation of the control signal depends on the internal state of the controller z. This general framework is able to capture both simple controllers [2], such as Proportional Integral and Derivative (PID) controllers, and optimal controllers, such as the Linear Quadratic Regulator (LQR). While the linear controller formalism is very general, it does not cover all the controller types, e.g., unless it is possible to find a closed-form solution for the control design problem, this formalism does not cover a Model Predictive Controller (MPC).

[4]Note that it is debatable whether the controller should calculate u_k or u_{k+1}. The formulation allows to handle both cases, but we are here assuming that the controller task adheres to the Logical Execution Time (LET) paradigm [11] and hence that it emits the output of the controller calculation at the end of the control period.

B. Computational Model

The controllers $\mathcal{C}$ should periodically calculate a control signal based on the control law presented in equation (2). A controller is generally implemented in a *control task*, i.e., periodic tasks with implicit deadlines. A typical implementation is the following.

```
while True:
  y = read_sensors()
  u, z = compute_control(y, z)
  sleep_until(next_activation)
  send_actuators(u)
```

Listing 1. Typical control algorithm execution.

The code in Listing 1 performs the following operations: (i) it samples the current plant measurements in `y` using the sensors (via the function `read_sensors`), (ii) it calculates and stores in memory the next control signal `u` and the controller's updated state `z` (via `compute_control`), (iii) it sleeps until the next activation (via `sleep_until`), and (iv) it sends the control commands to the actuators (via `send_actuators`).

For the control task, each iteration of the loop in Listing 1 is a new *job*, and the k-th iteration corresponds to the job j_k. The control job j_k is *released* at time $a_k = k\,T_s$, where T_s is the *period* of the control task. The objective of each job is to complete its execution before its corresponding *deadline* $d_k = a_k + T_s = (k+1)\,T_s$. We denote with f_k the time instant in which the control task *completes* the execution of job j_k. Ideally, $f_k \leq d_k$. If $f_k > d_k$, job j_k experiences an *overrun*, or a deadline miss.

Deadline misses can be caused by many different factors, like preemption from higher priority tasks and interrupts [19], timeouts due to long wait times on the sensor channel, and cache misses that introduce delays in accessing the controller's stored variables [25]. Regardless of what caused the overrun, the scheduler needs to react.

Indeed, two choices need to be made: what to do with the control signal that has not been calculated, and what to do with the task that missed its deadline. The most common approaches to select a control signal involve either *holding* the last received control command (i.e., $u_{k+1} = u_k$) or *zeroing* the output (i.e., $u_{k+1} = 0$). The choice of actuation mode is non-trivial and generally depends on the control system dynamic [18], [21].

For the task management, the literature [6] suggests three simple deadline overrun strategies: (i) *killing* the job that overran its deadline (and releasing a new one), rolling back any (possibly partial) change performed by the job that missed its deadline, thus reverting the internal task state variables to their original value, (ii) letting the job continue its execution,

skipping the subsequent job releases, until the current one has completed its execution, or (iii) combining the two, and letting the job continue its execution but at the same time *queueing* the subsequent job executions. In the case of skip and queue, the job that continues executing operates on outdated data. On the contrary, kill allows the task to always work with fresh data, with the risk of throwing away near-completed computations. One of our investigations showed that the queue strategy may create chain effects which can severely damage control systems [13], hence we only consider kill and skip as viable alternatives.

III. Deadline Miss Analysis

In the absence of deadline misses, the execution of a CPS is typically modelled as $\tilde{x}_{k+1} = \Phi\,\tilde{x}_k$, where $\tilde{x}_k$ is the closed-loop system state at iteration k. The matrix Φ is obtained aggregating the dynamic equations that determine the evolution of digital and physical quantities (i.e., the variables v for which we have specified an equation that determines v_{k+1} as a function of v_k). The matrix Φ thus includes the evolution of both the states of the physical and of the digital part of the system, $\tilde{x}_k = [x_k\ z_k\ u_k]^T$ and depends on the matrices A, B, C, D, F, G, H and K introduced in equations (1) and (2). In the case discussed in the previous section,

$$\begin{bmatrix} x_{k+1} \\ z_{k+1} \\ u_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 & B \\ G\,C & F & G\,D \\ K\,C & H & K\,D \end{bmatrix}}_{\Phi \text{ for a deadline hit}} \begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix}. \tag{3}$$

To assess if a CPS can endure deadline misses, we need the system evolution both when we experience a deadline hit and when we miss our deadline, obtaining two different matrices in place of Φ. The sequence of outcomes (deadline hits and misses) form a bit string, that we can analyse and constrain to follow prescribed patterns, typically following models like the weakly-hard task model [3].[5] We now want to answer questions like: if the controller does not miss more than three consecutive deadlines, is the system still stable, or, if in every five consecutive activations, we ensure that three jobs will successfully complete, can we rely on it?

The case in which the system cannot miss more than three consecutive deadline misses is a good initial example to show how the analysis tool works. We denote with Φ_0 the evolution of the system when there is a deadline miss. The matrix depends on the specific strategy that is used to handle the deadline miss. For example, if the job that experiences the deadline miss is killed and its changes are never saved, then when there is a deadline miss $z_{k+1} = z_k$. One may decide to keep the previous control signal valid in case of a deadline miss, i.e., $u_{k+1} = u_k$. This gives us the following equation for the deadline miss case.

$$\begin{bmatrix} x_{k+1} \\ z_{k+1} \\ u_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}}_{\Phi_0} \begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix} \tag{4}$$

Different alternatives are also possible, resulting in more complex Φ_0 (and Φ) matrices [13]. Imposing that there are no more than three consecutive deadline misses implies choosing, for the system evolution, matrices in the set $\Sigma = \{\Phi, \Phi\,\Phi_0, \Phi\,\Phi_0^2, \Phi\,\Phi_0^3\} = \{\Phi\Phi_0^d\}$ where $d \in \mathbb{N}$ and $0 \leq d \leq 3$.

Generally speaking, the stability analysis of a switching system is obtained by leveraging the *Joint Spectral Radius* [17]. Given $m \in \mathbb{N}, m > 0$ and a set of matrices $\Sigma = \{\Phi_1, \ldots, \Phi_m\} \subseteq \mathbb{R}^{n\times n}$, under the hypothesis of arbitrary switching over any sequence $s = < a_1, a_2, \cdots >$ of indices of matrices in Σ, the joint spectral radius of Σ is defined by:

$$\rho(\Sigma) = \lim_{N\to\infty} \max_{s\in\{1,\ldots,m\}^N} ||\,\Phi_{a_N} \cdots \Phi_{a_2}\Phi_{a_1}\,||^{\frac{1}{N}}. \tag{5}$$

The number $\rho(\Sigma)$ characterises the maximal asymptotic growth rate of matrix products from Σ (thus $\rho(\Sigma) < 1$ means that the system is asymptotically stable), and is independent of the norm $||\cdot||$ used in (5).

To analyse the case in which there are no more than three consecutive deadline misses, we should simply calculate the joint spectral radius of the set $\Sigma = \{\Phi\Phi_0^d\}$ where $d \in \mathbb{N}$ and $0 \leq d \leq 3$. However, the computation of the joint spectral radius is generally challenging, and has been a subject of extensive research in applied mathematics. One of the key results obtained by Blondel [4] is the proof that determining whether the joint spectral radius of a set of two square matrices is less than 1 is undecidable. This result is particularly striking as it applies even to matrices with relatively simple structures. However, it is possible to bound the joint spectral radius from below and from above, effectively obtaining bounds that will allow us to deem our CPSs safe. Over the years, researchers have developed various algorithms and approaches to approximate the joint spectral radius, such as [4], [5], [15], [14] and a library for the joint spectral radius approximation in MATLAB [20]. In [13] we have used these bounds in order to solve the problem of assessing the stability of CPS under a given number of maximum consecutive deadline misses (i.e., the **`RowMiss`** weakly-hard constraint), applying our method to the real case study of an electric motor.

Constraints that are more complex than the **`RowMiss`** constraint described above are harder to handle, as one needs to find the set of matrices that corresponds to all the possible system evolutions. We have leveraged the theoretical results to provide an analysis pipeline that allows us to determine,

[5] Constraints on the bit strings prescribe allowable limits for the manifestation of failures. For instance, a weakly-hard constraint λ_1 can mandate that the controller must be able to calculate a control signal within specific time limits except for a maximum of more than two failures in every three hundred consecutive iterations. Similarly, another weakly-hard constraint λ_2 may specify that the network should drop no more than two jobs in every ten consecutive activations. The two mentioned constraints are of type **`AnyMiss`**. The weakly-hard model [3] also defines **`AnyHit`**, **`RowMiss`**, and **`RowHit`** constraints. **`AnyMiss`**(m, w) prescribes that in a window of w consecutive activations there cannot be more than m misses. **`AnyHit`**(h, w) prescribes that in a window of w consecutive activations there must be at least h deadline hits. The **`RowMiss`**(m, w) and **`RowHit`**(h, w) constraints prescribe a maximum number of consecutive misses m or a minimum number of consecutive hits h.

given a weakly-hard constraint, if the corresponding switching system is asymptotically stable [23], [22]. This allows us to determine if the system is able to guarantee the deadlines (when they satisfy the pattern determined by the weakly-hard constraint).

When the switching sequences between the dynamics of Σ are not arbitrary, but constrained by a graph, the so called *constrained joint spectral radius* (CJSR) [8] can be applied. We leveraged the mathematical correspondence between weakly-hard constraints and specific regular languages. We generate the minimal finite state machine G that corresponds to a given weakly-hard constraint or to a set thereof [22].[6] Introducing $S_N(G)$ as the set of all possible switching sequences s of length N that satisfy the constraints represented by the finite state machine G, the CJSR of Σ is defined by

$$\rho\left(\Sigma, G\right) = \lim_{N\to\infty} \max_{s\in S_N(G)} || \Phi_{a_N} \cdots \Phi_{a_2} \Phi_{a_1} ||^{\frac{1}{N}}. \quad (6)$$

In general, computing or approximating the CJSR is harder than using the JSR. In [16], the authors propose a multinorm-based method to approximate with arbitrary accuracy the CJSR. Other works [12], [26] propose the creation of an arbitrary switching system such that its JSR is equal to the CJSR of the original system, based on a Kronecker lifting method. This is also the method that we use in [23] to analyse the system subject to deadline misses.

IV. Open Challenges

We proposed techniques to address deadline misses [13], [22] that analyse the bit strings corresponding to the CPS evolution, where a zero indicates a deadline miss and a one the correct timing execution of the control task. We exploited a link between weakly-hard constraints [3] and regular languages, generating the minimal finite state machine that corresponds to a weakly-hard constraint [23]. Using the final state machine and knowledge about the system evolution, we construct A_0 that represents the CPS dynamics during failures, and A_1, that models successes. We can then reduce stability assessment [24], [22] to calculating the joint spectral radius [17] of the set $\Sigma = \{A_0, A_1\}$.

However, scalability issues arise, as illustrated in Figure 1. The size of matrices in A_0 and A_1 in fact depends on: (i) the physical system (i.e., the size of Φ), (ii) the size of the finite state machine that encodes the weakly-hard constraints. Real systems fail rarely [3], [1]. To be useful, a verification procedure should be able to handle constraints such as $\lambda_1 = \textbf{AnyMiss}(2, 300)$–allowing up to 2 failures in 300 consecutive iterations [3], [13]. A constraint like this leads to complex (yet minimal) finite state machines, like M_{λ_1} with 44850 states, challenging computational limits and requiring extensive resources. To address this limitation, we can use more conservative constraints, like $\lambda_2 = \textbf{AnyMiss}(2, 10)$, which leads to M_{λ_2} with just 45 states, making the problem tractable. For constraints of tractable size (found for example via binary search), typically *CPSs are deemed unsafe*, which is inconclusive for the real-world scenario.

[6]The implementation of our analysis tool is open source and available at https://github.com/NilsVreman/WeaklyHard.jl.

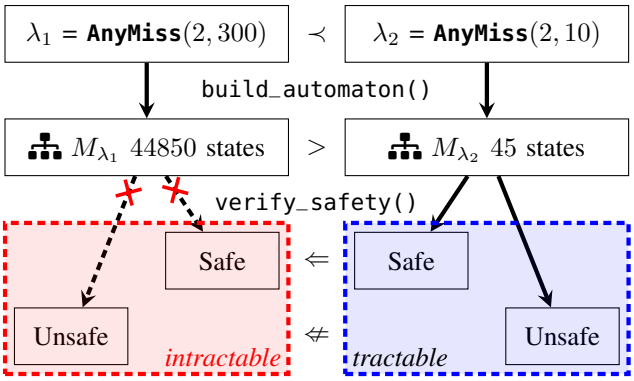

Fig. 1. Illustration of the scalability problem: λ_1 is a realistic constraint, but produces a finite state machine whose number of states exceeds the calculation limits for the joint spectral radius upper bound. On the contrary, the state machine for λ_2 has fewer states. Safety under λ_2 implies safety under λ_1, but introduces a significant amount of conservativeness. If the verification under λ_2 deems the system unsafe, the result is inconclusive for λ_1.

References

[1] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. An empirical survey-based study into industry practice in real-time systems. In *IEEE Real-Time Systems Symposium*, pages 3–11. IEEE, 2020.

[2] K. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design, Third Edition*. Dover Books on Electrical Engineering. Dover Publications, 2011.

[3] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, 2001.

[4] V. D. Blondel and Y. Nesterov. Computationally efficient approximations of the joint spectral radius. *SIAM Journal on Matrix Analysis and Applications*, 27(1):256–272, 2005.

[5] V. D. Blondel, Y. Nesterov, and J. Theys. On the accuracy of the ellipsoid norm approximation of the joint spectral radius. *Linear Algebra and its Applications*, 394:91–107, 2005.

[6] A. Cervin. Analysis of overrun strategies in periodic control tasks. *IFAC Proceedings Volumes*, 38(1):219–224, 2005.

[7] S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity and robustness of programs. *Communications of the ACM*, 55(8):107–115, 2012.

[8] X. Dai. A gel'fand-type spectral radius formula and stability of linear constrained switching systems. *Linear Algebra and Applications*, 2012.

[9] J. C. Dauer, B. Finkbeiner, and S. Schirmer. Monitoring with verified guarantees. In *International Conference on Runtime Verification*, volume 12974 of *Lecture Notes in Computer Science*, pages 62–80. Springer, 2021.

[10] G. E. Dullerud and F. Paganini. *A Course in Robust Control Theory*. Spinger New York, 2000.

[11] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1), 2003.

[12] V. Kozyakin. The berger–wang formula for the markovian joint spectral radius. *Linear Algebra and its Applications*, 448, 2014.

[13] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein. Control-system stability under consecutive deadline misses constraints. In *Euromicro Conference on Real-Time Systems*, volume 165 of *LIPIcs*, pages 21:1–21:24. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[14] C. Möller and U. Reif. A tree-based approach to joint spectral radius determination. *Linear Algebra and its Applications*, 463:154–170, 2014.

[15] P. A. Parrilo and A. Jadbabaie. Approximation of the joint spectral radius using sum of squares. *Linear Algebra and its Applications*, 428(10):2385–2402, 2008.

[16] M. Philippe, R. Essick, G. Dullerud, and R. Jungers. Stability of discrete-time switching systems with constrained switching sequences. *Automatica*, 72, 2016.

[17] G. C. Rota and G. Strang. A note on the joint spectral radius. In *Proceedings of the Netherlands Academy*, volume 22, pages 379–381, 1960.

[18] L. Schenato. To zero or to hold control inputs with lossy links? *IEEE Transactions on Automatic Control*, 54(5), 2009.

[19] J. A. Stankovic, M. Spurim, M. Di Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, 1995.

[20] G. Vankeerberghen, J. Hendrickx, and R. M. Jungers. JSR: a toolbox to compute the joint spectral radius. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, pages 151–156, New York, NY, USA, 2014. Association for Computing Machinery.

[21] N. Vreman, A. Cervin, and M. Maggio. Stability and performance analysis of control systems subject to bursts of deadline misses. In *Euromicro Conference on Real-Time Systems*, volume 196 of *LIPIcs*, pages 15:1–15:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[22] N. Vreman, R. Pates, and M. Maggio. Weaklyhard.jl: Scalable analysis of weakly-hard constraints. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 228–240. IEEE, 2022.

[23] N. Vreman, P. Pazzaglia, V. Magron, J. Wang, and M. Maggio. Stability of linear systems under extended weakly-hard constraints. *IEEE Control Systems Letters*, 6:2900–2905, 2022.

[24] J. Wang, M. Maggio, and V. Magron. SparseJSR: A fast algorithm to compute joint spectral radius via sparse SOS decompositions. In *American Control Conference*, pages 2254–2259. IEEE, 2021.

[25] W. Wang, P. Mishra, and A. Gordon-Ross. Dynamic cache reconfiguration for soft real-time systems. *ACM Transactions on Embedded Computer Systems*, 11(2), 2012.

[26] X. Xu and B. Acikmese. Approximation of the constrained joint spectral radius via algebraic lifting. *Transactions on Automatic Control*, 2020.

[27] K. Zhou and J. C. Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.