Analysis of Embedded Controllers Subject to Computational Overruns

Vreman, Nils

2023

Document Version:
Publisher's PDF, also known as Version of record

Analysis of Embedded Controllers Subject to Computational Overruns

Nils Vreman

LUND
UNIVERSITY

Department of Automatic Control

PhD Thesis TFRT-1141
ISBN 978-91-8039-688-2 (print)
ISBN 978-91-8039-687-5 (web)
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2023

Abstract

Microcontrollers have become an integral part of modern everyday embedded systems, such as smart bikes, cars, and drones. Typically, microcontrollers operate under real-time constraints, which require the timely execution of programs on the resource-constrained hardware. As embedded systems are becoming increasingly more complex, microcontrollers run the risk of violating their timing constraints, i.e., overrunning the program deadlines. Breaking these constraints can cause severe damage to both the embedded system and the humans interacting with the device. Therefore, it is crucial to analyse embedded systems properly to ensure that they do not pose any significant danger if the microcontroller overruns a few deadlines.

However, there are very few tools available for assessing the safety and performance of embedded control systems when considering the implementation of the microcontroller. This thesis aims to fill this gap in the literature by presenting five papers on the analysis of embedded controllers subject to computational overruns. Details about the real-time operating system's implementation are included into the analysis, such as what happens to the controller's internal state representation when the timing constraints are violated. The contribution includes theoretical and computational tools for analysing the embedded system's stability, performance, and real-time properties.

The embedded controller is analysed under three different types of timing violations: blackout events (when no control computation is completed during long periods), weakly-hard constraints (when the number of deadline overruns is constrained over a window), and stochastic overruns (when violations of timing constraints are governed by a probabilistic process). These scenarios are combined with different implementation policies to reduce the gap between the analysis and its practical applicability. The analyses are further validated with a comprehensive experimental campaign performed on both a set of physical processes and multiple simulations.

In conclusion, the findings of this thesis reveal that the effect deadline overruns have on the embedded system heavily depends the implementation details and the system's dynamics. Additionally, the stability analysis of embedded controllers subject to deadline overruns is typically conservative, implying that additional insights can be gained by also analysing the system's performance.

Acknowledgements

I would like to begin by expressing my deepest gratitude to everyone who has contributed to my journey towards a PhD, both professionally and personally. While it is impossible to mention everyone by name, I am immensely grateful to those who have brightened my days and helped me along the way.

Firstly, I owe a special thanks to the Department of Automatic Control and the many individuals who have made it such a welcoming and supportive workspace. Were it not for the people there, I would never have pursued a PhD. A special mention goes out to: Gautham, Mattias, Bagge, Martinka, Marcus TA, Marcus G, Pauline, Victor, Alex, Julian, Max NC, Johanna W, Frida N, Luka, and the Innebandy Crew. Another important reason why the department is such a pleasant workspace are the tireless efforts of Eva, Mika, Cecilia, Monika, Anders B, and Anders N. Thank you for providing invaluable support to all the PhD students passing through the department, all while spreading joy.

A special thank you is extended to my supervisor and dear friend, Martina. Your constant support, both professionally and personally, has been invaluable to me, and I would not be the person I am today without you. Words cannot express my gratitude, but please know that I will be forever grateful for having shared this journey with you. And to my co-supervisor and mentor, Anton, your encouragement during times of doubt has been instrumental in me reaching my goal. Thank you for all the beers shared, and to many more.

I am fortunate to count many of my colleagues as close friends. Claudio, thank you for sticking with me through thick and thin, and for all the unforgettable work discussions, life events, drinking nights, and travels. Ylva, I am thankful for your encouragement, your friendship, and the many meaningful conversations we have had (and will have) over dinners. Albin, Martin H, and Martin M, you have all been an inspiration (even in the face of adversity), and I am grateful for the laughter, dinners, pub nights, and game nights we have shared, and to the ones to come. Paolo, your kindness and knowledge have helped guide me, and I am lucky to have had the pleasure of getting to know you. Johan R, thank you for all the pub nights and trips we have enjoyed together. And finally, to Richard, thank you for for always having my back and for the joy you bring.

Last but not least, I am forever grateful for the love and support of my friends and family outside of work who have encouraged me throughout my journey towards a PhD. Calle, you have always been there for me, whether we were living together or not, and I appreciate all the new and cool things you have taught me, and the good times we have shared. To Olof and Erik B, despite us seeing each other far too rarely, I cannot thank you enough for your friendship. And to my parents, who may not always have understood the details of my work, but have always stood by me with unwavering support and love; thank you! To my brother Kalle, you have been a constant inspiration to me, and I am so lucky to have you in my life. And finally, to Elin, there are no words to thank you enough, and to Åse and Micke, your support means the world to me.

Financial Support The author is a member of the ELLIIT Strategic Research Area at Lund University. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Number 871259 (ADMORPH project). This publication reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Contents

Nomenclature

Notation	Description
$\mathbb{R}$	Set of real numbers.
$\mathbb{N}_{\geq}$	Set of natural numbers (inclusive), i.e., $0, 1, 2, \ldots$
$\mathbb{N}_{>}$	Set of natural numbers (exclusive), i.e., $1, 2, 3, \ldots$
$\mathbb{E}[\cdot]$	Expected value of a stochastic function.
$\lvert\cdot\rvert$	Cardinality of a set *and* absolute value of a number.
$\lVert\cdot\rVert$	Norm of a function *or* matrix.
$\lceil\cdot\rceil$	Ceiling function.
$\lfloor\cdot\rfloor$	Floor function.
$\mathcal{P}$	Plant representation.
$\mathcal{C}$	Controller representation.
$\mathcal{S}_{\text{cl}}$	Closed-loop system representation.
Φ	Closed-loop system matrix.
Γ	Closed-loop input matrix.
$\tilde{x}_k$	Closed-loop state vector at discrete time step k.
P_k	Closed-loop state vector's covariance matrix at discrete time step k.
J_k	Quadratic cost at discrete time step k.
τ	Arbitrary task in the real-time operating system.
$\mathcal{H}$	Deadline handling strategy.
λ	Weakly-hard constraint.
Λ	Set of weakly-hard constraints.
Σ	Alphabet.
$\mathcal{S}(\lambda)$	Satisfaction set of a weakly-hard constraint λ.
$\binom{x}{k}$	The `AnyHit` weakly-hard constraint.
$\overline{\binom{x}{k}}$	The `AnyMiss` weakly-hard constraint.
$\left\langle {x \atop k} \right\rangle$	The `RowHit` weakly-hard constraint.
$\overline{\left\langle {x \atop k} \right\rangle}$	The `RowMiss` weakly-hard constraint.
$\mathcal{G}$	Graph representation of the state machine describing a task's execution.
V	Set of vertices/states in graph $\mathcal{G}$ or in a Markov chain.
E	Set of labeled edges/transitions between vertices in V.

1

Introduction

Entering the digital age has forever changed how we interact with the world and how it interacts with us. Unlike only 20 years ago, from the moment we wake up in the morning till the moment we close our eyes at night, we interact with advanced computer systems. Our cellphones, work computers, and even our cars contain many computational devices, performing everything from menial tasks, such as checking the weather and accessing mail clients, to safety critical tasks, such as the car's ABS breaks and most of the engine's functionality. To put the digital growth rate in perspective, the semiconductor market share has more than quadrupled over the last 20 years [WSTS; SIA, 2022].[1]

Not only is the number of computational devices increasing, but their independent capabilities, functionalities, and complexities are growing steadily, all while the cost to buy and manufacture them has become cheaper. Obviously, the increased efficiency and reduced cost opened up new businesses and domains, in particular within the IT-domain, whilst also consolidating and automating preexisting industry. Integrating digital components and software solutions is nowadays the norm rather than the exception; this does not come as a surprise, considering that automating and simplifying the decision making and data collection yield both economical and safety benefits. Generally, integrating software into any domain help monitor system safety, log and transmit important data, orchestrate the execution of different components, and remotely micromanage system updates. Subsequently, software integration is a powerful tool that both improves efficiency and increases revenue, assuming everything behaves as intended.

Interconnecting multiple systems is, however, not a trivial task. As the systems are getting increasingly more complex, the surface for possible errors is also growing. After connecting two components together, new problems can be encountered in addition to the components individual faults; for instance, problems with the coupling or new problems in the individual components. A motorbike can experience all the same problems that a normal bike can encounter (such as a loose chain), but it can also experience issues from connecting the bike together with a motor (such as

[1] Semiconductors are components constituting the foundation of generally *all* electronic devices.

electric clutch). Similarly to the motorbike, systems relying on the interconnection of computational devices and digital components can experience complex coupling issues. For example, data transmissions can easily be delayed or stall indefinitely if data is lost, a computer's orchestrator can get overloaded, and systems with remote updates have the potential to break every time a new patch is installed. These problems are neither easy to detect nor troubleshoot; particularly since their origin can be obfuscated by complex software and hardware interconnections.

The effects of such errors can be extremely expensive and cause companies to lose billions of dollars. Typically the outcome of system faults is that the normal operation of the device (or machine) is degraded. The degradation can accumulate over time and either wear down the device or affect the end product. Obviously, there is a lot of money to be gained by extending the devices' lifetime through proper fault analysis. Furthermore, if the end product is inferior to the promised product, the consumers would go elsewhere — no matter whether the product is a service, such as cloud storage, or a physical product, such as a cellphone.

Arguably more important than the economic consequences are the risks to personal safety, security, and privacy. One of the modern era's most devastating examples are the Boeing's 737 MAX crashes, killing 346 people in two subsequent crashes.[2] The crashes were caused by erroneous sensor readings being misinterpreted by the flight control system, resulting in the planes nosediving into the ground. Another relevant (although less lethal) example is the infamous Stuxnet worm.[3] Stuxnet infiltrated the system controlling the gas centrifuges in multiple Iranian nuclear plants, significantly damaging them whilst also collecting critical information.

It is generally impossible to guarantee that today's complex computer systems are absolutely safe, secure, and performs according to specifications under all conditions. Additionally, testing for all possible future problems is expensive and time consuming at best and infeasible in practice. It is therefore crucial to develop easy-to-use, powerful tools to simplify the analysis of both the systems' performance and safety properties.

The purpose of this thesis is to provide tools and methods for analysing systems experiencing faults. In particular, the focus is to analyse software integrated systems where the faults occur in the interconnection between software and hardware. By treating accessibility, clarity, and generalisability as first-class citizens we aim to lower the threshold for using the powerful tools provided. More specifically, we provide tools to analyse real-time control system performance and stability when the real-time tasks governing the control computations are subject to deadline overruns. The following subsection introduces the basic context for the real-time control system constituting this thesis' principal theme.

[2] `https://en.wikipedia.org/wiki/Boeing_737_MAX_groundings`
[3] `https://en.wikipedia.org/wiki/Stuxnet`

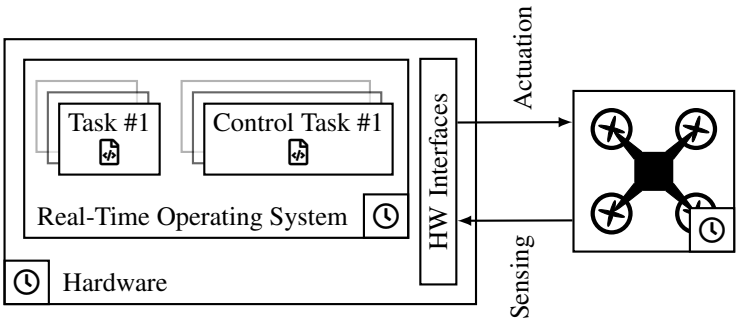


Figure 1.1 A control system represented at a high level of abstraction. The plant is represented on the right and its digital control structure is shown on the left. The control structure comprises hardware and its interfaces with the plant as well as a real-time operating system and its running tasks, among which are the control tasks.

1.1 Real-Time Control Systems

Fundamentally all systems today contain a certain level of automation, whether it is automatic heat control in buildings or memory allocation in the cloud for storing photos. The science of making systems automatically behave according to predefined specifications is called *automatic control*. Characteristic for many automatic control systems (*control systems* for short) is that they employ *feedback*, i.e., data collected from the system is routed back and used in the decision mechanisms to control the system. As an example, consider the temperature control in a room, if the actual temperature is known, it can be used (fed back) to determine whether the heating should be turned up or down to meet the desired temperature. A specific class of control systems are the *real-time control systems*, which are defined by guaranteeing the timely execution of software in the control system. A common misconception is that real-time systems are inherently very fast; however, the definition only relates the timeliness of the system to a precise notion of correctness. The real-time system's correctness is then expressed as guarantees that a set of predefined temporal constraints are met. To enforce the satisfaction of these constraints, a *real-time operating system* (RTOS) is typically employed.

A high-level abstraction of a real-time control system is depicted in Figure 1.1. Next, the individual components seen in the figure are introduced.

Plant The right part of the figure depict the process we are trying to control (denoted the *plant*). This could be anything from flight control of a drone (as in the figure), indoor heating systems, or the load on a server in a data centre. In this and following chapters, the drone's flight control system will be used as a recurring example to illustrate the different concepts. The arrows going to and from the plant indicate the flow of data; actuation data goes into the plant and sensor data is col-

lected from the plant. Actuation data refers to the commands sent to the components responsible for movement or change in the plant, i.e., the actuators. Similarly, the sensor data is all information collected by the sensors, e.g., the drone's acceleration in different directions or its angular velocity. These signals are transmitted via the hardware interfaces on the computational unit responsible for controlling the plant. Historically, these signals were transmitted via wire, but in the last couple of decades wireless communication has become more common [Park et al., 2018].

A communication protocol is used for the plant to communicate with the hardware interfaces.[4] The choice of protocol is domain dependent, for instance, in the automotive industry the controller area network (CAN) is widely used [Voss, 2005]. There exists a plethora of domain specific communication protocols, but some established ones include Profibus, Modbus, Ethernet/IP, and (the aforementioned) CAN.

Hardware Depending on the application, the hardware used to control the plant can be anything from a logic-based system (e.g., programmable logic controllers) to a general purpose computer (e.g., laptops or server systems). We mainly refer to *microcontrollers* (MCUs), i.e., small computers with integrated memory, central processing units (CPUs), graphical processing units (GPUs), and programmable input/output peripherals (PIOs) all on a single chip; however, we emphasise that the presented results are not bound to a specific hardware architecture. It is also common to connect multiple levels of control hardware together. For instance, having a high-level trajectory planner communicate with a low-level control structure whose objective is to enforce that the desired trajectory is followed, e.g., the hovering height of a drone.

The choice of computer architecture is often flexible and can be changed depending on the plant structure. It is for instance not uncommon that the plant sensors include their own MCU to perform data processing before transmitting it to the central control hardware [Karray et al., 2018]. Another common architecture choice is having many single objective, specialised MCU nodes operating together towards a global target, e.g., controlling the rotational velocity of the individual drone propellers during flight.

Clocks There exists another discrepancy between the components of the real-time control system: the time quantisation. The plant, hardware, and RTOS are most likely executing in different time scales, where the plant (at least in the drone example) is likely to execute in continuous time while the hardware and RTOS are both executing in (different) discrete quanta. Extremely simplified, the hardware contains a *clock* that measures the progress of physical time in *ticks* rather than seconds (like a wall-clock would). For the hardware clock, these ticks are physical

[4] A communication protocol is a set of rules setup in order for two or more actors in a network to be able to transmit information to one another. The rules include (but are not limited to) semantics, i.e., how to format the information, and synchronisation, i.e., how much and how fast the information is transmitted.

events (typically the oscillations of a crystal) occurring with a known frequency. The RTOS clock is then based on the hardware clock; it reads the hardware clock at a frequency specified by the user and wraps it in a virtual layer to improve applicability. The granularity of the RTOS clock is thus quite coarse, which in turn introduces *release jitter* and *execution time variations* on the tasks. Additionally, the inconsistencies between different clocks is a difficult problem and it has in fact warranted its own research domain, i.e., clock synchronisation.

Real-Time Operating System Commonly, real-time control systems rely on a real-time operating system to schedule and constraint the temporal execution of a set of *tasks*. Each task is assigned a *priority* (a value to describe how important the task is), a dedicated function to execute (e.g., compute a value, transmit data, or log data), and a *deadline* before which the function is supposed to complete its execution. Typically there exists more than one task executing in the RTOS. To guarantee that each task is assigned the correct amount of processor time, a *scheduler* is used to orchestrate the tasks' execution. Specifically, the scheduler (i) swaps tasks in and out, (ii) wakes up tasks that are currently not executing but should start executing, and (iii) interrupt tasks that are currently executing when something with higher priority requires the processor. The orchestration is based on a scheduling algorithm, where some of the most popular algorithms include: fixed-priority, earliest-deadline-first, and round-robin. Additionally, the tasks are not supposed to have any information about the RTOS orchestration and it is thus the RTOS responsibility to ensure that the task has access to its own *context*, i.e., the resources it requires.

The time it takes for a task to finish executing its corresponding function can vary greatly between iterations. For instance, if a task contains different conditional branches its execution time may be dependent on the branch taken.[5] To quantify a task's execution time, the simplest task models approximate it as the *worst-case execution time* (WCET), i.e., the maximum length of time the task could execute on the specific hardware. The WCET is typically pessimistic, but it is also important for guaranteeing reliability in safety-critical real-time systems.

Since there may exist many tasks in the RTOS and the scheduler can swap them in and out arbitrarily, there are no guarantees that a task will execute its entire function consecutively. Firstly, as tasks can be dependent of one another, one task may have to wait for another task to change its state (or compute a value) before completing its execution. Secondly, if the executing task gets *preempted* by the scheduler in favour of another higher-priority process (e.g., an interrupt or a higher-priority task), it will again have to wait for the scheduler to switch it back in. The maximum length of time from that a task starts executing until it finishes is called the *worst-case response time* (WCRT). If a task's WCRT is smaller than its deadline, the task will be guaranteed to always meet its deadline.

[5] If a function's behaviour change depending on a logical condition, it is said to have conditional branches.

Controller As can be seen in Figure 1.1, we conceptually distinguish *control tasks* from normal tasks, even though there is no discernible difference between these tasks from the RTOS perspective. The control tasks are all the tasks responsible for controlling the plant, while the normal tasks take care of everything else. Taking the drone as an example, the control tasks can be two tasks where one is making sure that the rotational velocity of each propeller is following the desired setpoint whilst the other is controlling the motors' relative speeds to propel the drone forwards. Additionally, there are other tasks communicating sensor data (e.g., barometric pressure, MCU temperature, acceleration, etc.) to the central processing unit.

As elaborated upon in Chapter 2, we partition the tasks in these two categories because we are specifically interested in analysing the control tasks' effect on the real-time control system when their temporal execution is unreliable. In particular, the aim of the thesis is to investigate the control system's behaviour when the control tasks overrun their respective deadlines. This is particularly relevant for control systems, because enforcing that the control task's WCRT is shorter than its deadline (i.e., that the task *never* overruns its deadline) would involve postponing the deadline. Holding off the deadline results in fewer control updates, hence severely degrading the control system's performance. Consequently, allowing a certain number of deadline overruns can improve the control system's overall performance.

Timeliness

In some situations, real-time control systems are executed under suboptimal timing conditions. As already mentioned, faults typically lead to reduced quality of the end product and can in the worst case be lethal. However, the term "timing faults" is an oversimplification of a complex class of problems. When analysing timing faults and irregularities it is thus important to properly define the fault type under analysis. The following list include some of the most general timing irregularities analysed in literature.

- *Computational overruns* – When there exists computational elements in the system, i.e., something that takes the current state of the system and transforms or translates it, there is the possibility that it will not complete its execution, either on time or at all. Generally, this is connected to a real-time system if the computation has to complete before a predetermined deadline.

- *Time delays* – Time delays might be some of the oldest timing problems analysed in the literature. Nowadays, time delays involve both internal time delays and input/output delays, i.e., respectively when the plant includes time delays or when the sensing, actuation, or control contains time delays. The control community has developed methods to both analyse and design controllers for systems with known time delays [Mirkin, 2004; Mirkin and Palmor, 2005].

- *Jitter* – Variations in the time delay are called jitter. Note that the previously mentioned release jitter, originating from the coarse granularity of the RTOS

clock, is just one type of jitter. Another type of jitter appear when transmitting data over a network; if the packet latency (time delay) is constant in time, there is no jitter in the system. Instead, if the latency varies over time (as it usually does), there exists jitter in the system. The jitter intensity depends on the size of the latency fluctuations. In the control literature, methods to compensate for jitter have been proposed [Cervin et al., 2004].

- *Communication losses* – If data packets transmitted over a network are either lost along their route or too delayed to be useful, they are considered *lost*. Another case when the packets are dropped is when the network is shut down or overloaded, either by too much traffic or an attack. Packet losses can occur both on the way to and from the hardware.

Conceptually, it may be natural to view some of the timing irregularities as equivalent from a system-wide perspective. Distinguishing time delays from jitter or communication losses from computation overruns is difficult in this context. For instance, both communicational losses and computational overruns affect the underlying software implementation; thus, it is only natural to assume that one can be substituted for the other when analysing the system in order to save both time and effort. However, the different models hold unique information that does not necessarily overlap. It is therefore crucial to analyse all the appropriate models in order to get a comprehensive picture of the specific system's behaviour.

This thesis specifically targets systems subject to computational overruns, and (to a minor extent) communication losses. Of particular interest are the problems that occur in real-time control systems due to control tasks overrunning their corresponding deadlines. It is convenient to blame computational overruns on poor system design and programming errors; however, even a flawless system design can experience overruns due to, for instance, cache memory misses [Gracioli et al., 2015] or radiation-induced faults [Tsog et al., 2021]. More commonly, it is known that the real-time control system's nominal performance is degraded if a design that completely avoids overruns is used, i.e., if the deadline is postponed until after the WCRT of the controller. Thus, with the awareness that it can cause transient faults, the control system engineers can choose to tolerate a few overruns when the control task's execution time is near its worst case, in order to improve the system's nominal performance.

1.2 Outline

This thesis is a collection of papers and is divided into two parts. The first part comprises the first three chapters and serves both as a summary and as an extension of the motivation for the research work. The second part includes the papers that constitute the major scientific contribution of this thesis.

The first chapter presented a high-level introduction to the relevant concepts and problems. Next, Chapter 2 provide a rigorous presentation of the concepts that were loosely introduced in Chapter 1. In addition to the proper problem description, the chapter also outlines the relevant background and related work from both the real-time and control theoretical domains. For each of the five papers constituting the main part of the thesis, Chapter 3 contains a short summary of its content, a brief description of its scientific contribution, and the respective authors' individual contribution.

2

Background

This chapter presents the necessary background and motivation for the remainder of the thesis. We divide the chapter in two primary parts. First, a discussion on the theoretical aspects of real-time systems is provided. An extended introduction to how real-time operating systems operates is presented, e.g., processor sharing, task states, scheduling strategies, etc. However, the main focus is dedicated to the most commonly used task models and their respective advantages and disadvantages, with respect to deadline overruns. Additionally, we provide a brief discussion on state-machine applicability to the aforementioned task models. Next, the relevant control theoretical background is presented based on the theory of real-time systems. Two different system modelling approaches are introduced: switching systems and Markov jump linear systems. Both models are particularly relevant for real-time systems where the control task can overrun its deadlines. Specifically for these systems, we present and discuss different stability and performance analyses.

2.1 Real-Time Systems

We begin with an introduction to real-time system fundamentals. The breadth of the topic prevents a comprehensive review of the existing literature to fit within the scope of this thesis. In fact, real-time systems are all information processing systems which reacts to external input within a predetermined deadline. This includes sensors, actuators, process control, machine vision, robotics, and health care systems, to acknowledge a fraction of all real-time systems. Instead, we focus the attention to the elements which impact real-time control systems the most, i.e., CPU provisioning, memory management, periodic tasks, task models, scheduling policies, and execution models. Since the RTOS is tightly interconnected with the hardware, it is natural to illustrate them jointly. Next, we describe the underlying hardware and real-time architecture seen in Figure 2.1.

Although this thesis does not discern different hardware architectures from one another, it is appropriate to talk about *microcontrollers* and *embedded systems*, in particular due to their prevalence in real-time control systems. Despite being two

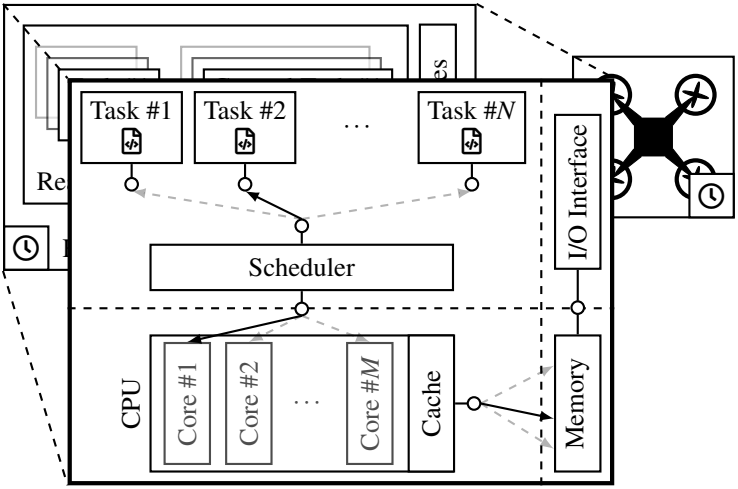


Figure 2.1 A more detailed view of the digital elements of the control system introduced in Figure 1.1. Processor, memory, and hardware interfaces are represented as well as the scheduler responsible for determining which task(s) that are currently executing on the hardware platform.

different hardware architectures, the terms embedded system and microcontroller will carelessly be used interchangeably due to their natural similarities. As introduced in Chapter 1, microcontrollers (MCUs) are small computers with integrated processors, memory, and I/O peripherals. Most embedded systems are based on microcontroller architectures, however, some embedded systems are based on one or more microprocessors with external memory and I/O peripherals. Hence, microcontrollers are embedded systems which can also be used to develop more complex embedded systems, but an embedded system is not necessarily a microcontroller.

The basis of every hardware architecture is a *central processing unit* (CPU, or simply *processor*). This is the electronic component responsible for executing the desired functions. Each function (or program) is translated into a list of instructions to be executed on the CPU. These instructions belong to the machine's language used to tell the processor what type of operation to execute, e.g., load a specific memory register or execute an arithmetic operation. The time it takes for the CPU to execute one instruction, i.e., fetching the instruction from memory before decoding and executing it, is typically called an *instruction cycle* (or simply *cycle*); this is the basic unit used to measure CPU speed. To execute the program instructions, the processor can contain one or more *cores*, respectively denoting the processor as *single-core* or *multi-core*. Each core is able to execute a list of program instructions. Hence, the advantage of using multi-core processors (compared to single-core processors) is the increased number of instructions that can be executed in paral-

lel. However, this gain comes at the cost of an elevated system complexity where the memory and application layout has to be adapted to the multi-core architecture [Brandenburg, 2011].

Integrated with the processor is a *cache* memory, i.e., a small but fast memory that is easy to access from the operational cores. The cache memory stores recently accessed instructions and data to reduce the latency induced by fetching from *main memory*, i.e., the main hardware storage. Most modern CPUs have a layered cache memory hierarchy, where the smallest and fastest layer is denoted L1, the second smallest and fastest is denoted L2, and so on. When the processor needs to access some data, it first examines whether the data exists in the cache and in that case fetch it from there; otherwise, it collects the data from the main memory.

If a task wants to access cached data (or instructions) that cannot be found, it is said to experience a *cache miss*; similarly, a *cache hit* occurs when the sought data is found in the cache. Cache misses can arise if:

- the size of the requested data is too large to fetch;
- the requested data is not yet loaded into the cache; or
- the data has been evicted from the cache, e.g., to make room for more recently retrieved data, or because the cache has been flushed due to security reasons.

Ideally, the number of cache misses that a task experiences is kept to a minimum, in particular since fetching data from the main memory can incur large timing overheads on the task execution. Additionally, the longer a task executes, the less likely it is to contract cache misses. Intuitively, the task will experience a few initial cache misses when the data is loaded into the cache, but thereafter the cache will be occupied by relevant data and the cache misses should decrease. This is also known as *cache warming*. If the task continues to experience significant cache misses even after the cache warming phase, it is said to be *thrashing* the cache, i.e., continuously experiencing cache misses. Thrashing can severely impact both real-time performance, energy consumption, and even collapse system execution [Wadleigh and Crawford, 2000]. In multi-core setups where different cores share a cache layer, thrashing is a big concern; however, there exists strategies to mitigate frequent interference from different tasks sharing the same cache [Brandenburg, 2011]. To help mitigate cache eviction (both in single- and multi-core setups), *cache partitioning* is typically employed. Cache partitioning reserves specific memory addresses for specific tasks, whilst reserving others for shared data. Thus, cache evictions are limited to the specific cache memory regions that are shared among multiple tasks, unless the cache is flushed due to security reasons.

To interface with the external environment, the hardware uses *input/output peripherals* (I/O peripherals). The peripherals are all external components connected to the hardware, e.g., sensors, actuators, or routers. Depending on the hardware, the peripherals can either be connected to the circuit board responsible for joining the

different components together or directly into the CPU. If the link to the external environment is wireless, the I/O peripherals are not necessarily sensors or actuators, but rather radio antennas, Bluetooth transmitters, or Wi-Fi routers (depending on the wireless communication protocol) interacting with the sensors and actuators. Typically, each peripheral is assigned to an *I/O port* in the device, i.e., a unique number to know which physical pin to transmit and receive data through.

Separately from the I/O port, the *communication protocol* defines the rules used to pack and unpack the packets sent between the peripherals and the hardware over the *communication channel*. As an analogy, consider a postcard being sent between England and France; the port is where we choose to send the letter, i.e., both the address to send it to and the postage stamp, while the protocol is the content of the message, i.e., the chosen communication format. Communication protocols and their implementation details belong to a vast research topic which falls outside the scope of this thesis. However, it is an important component of real-time networked control systems and is thus briefly introduced here.

Information transmitted over a network (wired or wireless) is generally represented as a set of bits (ones and zeroes) to be read in series or parallel; without loss of generality, we will only mention the serial case. The communication protocol defines the rules determining how the transmitter should encode its data in order for the receiver to decode it using the same set of rules. The rules are highly dependent on the communication protocol and its application domain. We illustrate the idea of communication protocols with an example: consider the case where a transmitter wants to send the character `R` using the universal asynchronous receiver-transmitter (UART) protocol.[1] The rules defined by this protocol states that each data packet contains exactly 8 bits of information, is prepended with a start bit (0), and is appended with a stop bit (1). Since the binary representation of `R` is 01010010 (using ASCII encoding), the encoded character's packet representation to be sent over the network is then

$$\underbrace{0}_{\text{Start bit}}\ \underbrace{01010010}_{\text{Data bits}}\ \underbrace{1}_{\text{End bit}}\ .$$

Transmitting a message, such as `RTS`, thus involve encoding each character individually before transmitting the encoded bit stream in sequence,

$$\underbrace{0010100101}_{\texttt{R}}\ \underbrace{0010101001}_{\texttt{T}}\ \underbrace{0010100111}_{\texttt{S}}.$$

The network over which the packets are transmitted, typically consist of one or more routers forwarding the packets between different target locations. To determine where to forward the packet to, the packet includes a *network address* that is read by the router before rerouting the packet according to a routing policy.[2] Additionally, each router contains a buffer to store incoming packets before processing

[1] Assuming ASCII encoding of the character, 8 data bits, no parity, and 1 stop bit, i.e., UART 8-N-1.

[2] The network address is an identifier to help recognise where to forward the packet to. Common examples of network addresses are IP addresses and MAC addresses.

them. Processing the packets in the buffer is efficient, but it requires some non-negligible overhead, i.e., reading the network address and deciding where to forward the packet. Thus, under normal conditions the receiver will experience packet latency and jitter, but if the network traffic is heavy the buffer space can quickly be exhausted. In other words, if the packets arrive faster than what the router can process it becomes congested.

Multiple policies have been developed to control the congestion, one common example being the tail drop policy [Comer, 2013]. Generally all the congestion control strategies employ intentional *packet dropping*, i.e., if a packet arrives while the buffer space is exhausted, the congestion controller will remove either the arriving packet or one in the buffer (depending on the strategy). In addition to the congestion controller dropping packets, there are intermittent packet drops due to, e.g., packets being misrouted [Bradley et al., 1998], security threats [Hansman and Hunt, 2005], software bugs [Mai et al., 2011], or wireless communication [Zhu et al., 2021].

Regardless of the packet drops' origins, they can have dire consequences for real-time control systems. Losing packets on the network connecting the control hardware and the plant, is the same as losing sensor measurements or control commands. Generally, this will degrade the control performance [Nilsson, 1998], however, if enough packets are lost it could cause critical system failures or crashes [Xiong and Lam, 2007].

To simplify the interface with the hardware while guaranteeing timeliness, a real-time operating system is commonly employed. The RTOS is responsible for orchestrating the tasks' execution and allocating resources (e.g., memory and CPU time) to said tasks. The terms "task", "process", and "thread" are frequently confused in many documents; to avoid this confusion we provide a definition in the context of real-time operating systems:

- *Process* – A process is a computer program with its own stack, control block[3], and instruction set.
- *Thread* – A thread is an entity within a process that share the memory context with additional threads.
- *Task* – The term *task* is used analogously with *process*.

The notational confusion likely arose from the *multithreading*, *multiprocessing*, and *multitasking* paradigms. In a multiprocessing environment, multiple tasks can execute concurrently, each on a separate core; multithreading is a CPU feature for executing multiple threads concurrently on a single core; and, multitasking is when a single core is executing multiple tasks concurrently.

[3] A task's control block (TCB) includes descriptive information about the task, for instance, the identifier used by the scheduler, its priority, and the task's state (running, ready, blocked, suspended, etc.).

The RTOS typically employ a scheduler to orchestrate the tasks and to provide them with the appropriate resources. The scheduler is responsible for switching tasks in and out, making sure that the correct task context (the task's resources) is brought into scope, handling interrupts, and ensuring fairness among the entities sharing the resources, e.g., interrupts, tasks, and kernel methods. How the scheduler assigns resources is decided by the scheduling algorithm. Most scheduling algorithms adopt a *preemptive* approach to assigning resources, i.e., the scheduler is run once every time slice (time quanta) to choose which task to switch in. Classical examples of preemptive scheduling algorithms are: (i) fixed priority preemptive (FPP), where the task with the highest predetermined priority value is executed; (ii) earliest deadline first (EDF), where the task with the shortest time to its corresponding deadline is executed; and (iii) round-robin (RR), where the tasks are switched into scope in a circular order. Depending on the choice of algorithm, the real-time system's execution pattern may vastly differ. For instance, two different scheduling strategies applied to one set of real-time tasks, may or may not result in the system being *schedulable*, i.e., all the temporal constraints are satisfied. If the RTOS tasks are not schedulable there exists tasks overrunning their corresponding deadlines.

In order for the scheduler to know when to stop the currently running task in favour of switching in another, the RTOS clock triggers an *interrupt* at every clock tick. Interrupts can come from both hardware and software, but the RTOS ticks are triggered by the software clock in the RTOS.[4] Attached to the interrupt is generally an *interrupt service routine* (ISR), i.e., a callback function to execute when the specific interrupt is triggered. For instance, in the context of the scheduler's tick interrupt, the ISR may be responsible for switching out the currently active task before switching in a new task and its relevant context. The ISR connected to the RTOS clock tick is also one of the major causes for release jitter in RTOS, i.e., the time it takes to put the suspended task into the ready queue before picking a new candidate to execute varies between invocations. Since the RTOS suspends the execution of the active task whenever an interrupt is triggered (even if it happens in the middle of a time slice), it is crucial that the ISR is fast, to avoid stalling the processor. Hence, if the callback function takes too long to execute or if the ISR is triggered too often, it can cause significant time delays in the schedule execution.

Employing a scheduler for single-core processors follow the described paradigm, however, for multi-core systems additional design choices have to be made. Fundamentally, two different approaches have been taken to scheduling to a multi-core processor: *global* or *partitioned* scheduling. The former assumes one scheduler responsible for scheduling all the tasks in a global ready queue to the individual cores based on available and required resources. On the other hand, partitioned scheduling (sometimes referred to as *clustered* scheduling) involves

[4] Hardware interrupts come from events changing the state of the system, e.g., external signals triggering that they need attention from the RTOS, watchdog timers triggering an interrupt at set time intervals, or spurious interrupts (electrical anomalies) [Drepper and Molnar, 2003].

dividing the tasks into partitions that are then mapped to separate cores with individual schedulers. Despite having additional computational resources to work with, multi-core systems are subject to deadline overruns similarly to single-core systems, when the capacity is exceeded or due to other events such as deadlocks and locking of shared resources.

Tasks All the real-time tasks considered in this thesis are *recurrent*, i.e., they do not terminate during system operation. The recurrent task model simplifies the a priori analysis of the real-time workload's effect on the system execution. A plethora of methods have been derived to model the recurrent task execution, ranging in complexity from the classic Liu and Layland model [Liu and Layland, 1973] to directed acyclic graph (DAG) models [Saifullah et al., 2014]. Henceforth, the terms *recurrent tasks* and *tasks* will be used analogously.

The task model adopted in this thesis defines a task τ as a sequence of *jobs* j_k, where each job is responsible for executing one full iteration of the task's function. Here, k counts the number of discrete time steps since the task was created. Each task τ_i is characterised by the triplet (e_i, d_i, p_i). Here, for each job; $e_i > 0$ is the *worst-case execution time* (WCET); $d_i > 0$ is the relative *deadline*; and, $p_i \geq d_i$ is the minimum interarrival *period*. The RTOS scheduler releases a job j_k at time a_k (the job's *release time*) and the job then completes its execution at time f_k (the job's *completion time*). A recurrent task is *periodic* if its jobs are released at equidistant time points, i.e., $a_k = k \cdot T$ where T is fixed. In particular, control tasks are typically implemented as periodic tasks, hence one job is released in every period. To make sure that the periodic task's jobs finish their execution before the subsequent job is released, it is common to adopt *implicit deadlines*, i.e., that job j_k completes its execution before the release time of job j_{k+1}; formally, it implies that $f_k \leq a_{k+1} = k \cdot T + T = a_k + T$ or simpler $d_i = p_i = T$.

Under ideal computational conditions, each job completes its execution before its corresponding deadline, i.e., $f_k \leq a_k + d_i$. However, it can happen that the individual job has not yet finished executing when it reaches the end of its allotted time budget. We then say that the job experiences a *deadline overrun* (also referred to as *deadline miss* or *computational overrun*). Respectively, if the job completes before its deadline, it *meets* its deadline (experiences a *deadline hit*).

DEFINITION 1—DEADLINE OVERRUN
The k-th job (j_k) of a task τ_i is said to experience a deadline overrun if

$$f_k > a_k + d_i.$$

Computational overruns are present in generally all real-time domains from avionics and defence to consumer electronics [Åkesson et al., 2020], thus highlighting the importance of analysing their impact on the systems' functional correctness.

Every real-time system behaves differently in the presence of deadline overruns. Depending on the application, some systems crash while others experience a

degraded efficiency. Due to this individuality, most real-time systems were historically divided into two classes describing how the systems were affected by computational overruns.

- *Hard real-time systems* – It is imperative that all deadlines are met in order to prevent critical system failure.

- *Soft real-time systems* – The perceived quality of the system is degraded with the number of overrun deadlines, but it is unlikely that it will impact system safety.

Soft real-time systems typically do not crash catastrophically when they experience a finite number of deadline overruns. To analyse these systems, their deadline overruns are typically modelled using stochastic processes. For instance, the most basic models assume that jobs' deadline outcomes are independent and identically distributed random variables, i.e., that the outcome of each job depend only on the probability that this specific job overruns its deadline. Intuitively, this assumption is too simplistic for real systems and instead stochastic models for the task execution were developed based on, for instance, Markov chains [Liu et al., 2005; Friebe, 2022; Abeni et al., 2017; Lincoln and Cervin, 2002] and task chains [Manolache et al., 2004; Liu and Anderson, 2010], where the transitions between states are probabilistic. Despite the hard and soft classes covering many real-time systems, they do not cover all cases. In particular, embedded controllers are typically better described using the *firm real-time system* model, characterised by being able to overrun a few, but not too many, deadlines before causing critical system failure.

Arguably the most recognised firm model is the *weakly-hard* task model [Bernat et al., 2001]. These models were originally devised to provide formal guarantees to tasks that can tolerate occasional deadline overruns, e.g., control tasks where decreasing the sampling time would improve the overall performance whilst introducing intermittent computational overruns. What defines a weakly-hard task is that the distribution of deadline hits and misses during a window of k jobs is precisely bounded.[5] In other words, in addition to the number of overrun deadlines that a task experiences in a window, the sequence in which they appear is also affecting the task execution. We here compile the definitions of the weakly-hard models:

DEFINITION 2—WEAKLY-HARD TASK
A weakly-hard task τ is a task that satisfies (at least) one of the following constraints:

(i) $\tau \vdash \binom{x}{k}$ (`AnyHit`)*: in any window of k consecutive jobs, the minimum number of deadline hits is x;*

[5] From context it will always be clear whether k is used to respectively denote the window length of a weakly-hard constraint or to count the discrete number of job iterations of a recurrent task.

(ii) $\tau \vdash \left\langle {x \atop k} \right\rangle$ (**RowHit**)*: in any window of k consecutive jobs, the minimum number of consecutive deadline hits is x;*

(iii) $\tau \vdash \overline{\left({x \atop k} \right)}$ (**AnyMiss**)*: in any window of k consecutive jobs, the maximum number of deadline misses is x; and*

(iv) $\tau \vdash \overline{\left\langle {x \atop k} \right\rangle}$ (**RowMiss**)*: in any window of k consecutive jobs, the maximum number of consecutive deadline misses is x;*

for some values of $x \in \mathbb{N}_{\geq}$, $k \in \mathbb{N}_{>}$, *where* $x \leq k$.

Here, the $\vdash$ symbol is used to indicate that all possible sequences of deadline hits and misses of τ satisfy the right hand side.

To formalise which sequences of deadline hits and misses that are permitted under a specific weakly-hard constraint, an alphabet is introduced. For historical reasons, the language used to characterise the deadline outcomes of a weakly-hard task is binary, i.e., it consists solely of two unique character mappings to a deadline hit and a deadline miss.[6] Formally, the *alphabet* of outcomes is denoted $\Sigma = \{0,1\}$, where 0 indicates a job overrunning its corresponding deadline and 1 represents a job meeting its deadline. With the use of the alphabet and conventional language theoretical notation [Hopcroft et al., 2006], a *character* $c_k \in \Sigma$ is defined as the outcome of the k-th job. Similarly, a *word* w is a sequence of characters, i.e., $w = \langle c_1, c_2, \ldots \rangle$. Hence, a word is representing a sequence of deadline hits and misses. The set of all all length N words that can be constructed from the alphabet Σ is denoted Σ^N.

Since all of the weakly-hard constraints act on the same language it is natural to ask whether they are relatable to one another, or not. In [Bernat et al., 2001], the authors show that the constraints are in fact comparable using the sets containing all sequences satisfying the specific constraints. With a slight abuse of notation we will let $w \vdash \lambda$ represent the case when a word w (outcome sequence) satisfies the weakly-hard constraint λ.

DEFINITION 3—SATISFACTION SET
The satisfaction set $\mathcal{S}_N(\lambda)$ *of an arbitrary weakly-hard constraint* λ*, is the set of all length* $N \in \mathbb{N}_{>}$ *words w satisfying* λ*. Formally,*

$$\mathcal{S}_N(\lambda) = \left\{ w \mid w \in \Sigma^N, w \vdash \lambda \right\}.$$

To simplify notation, the set of infinite length words satisfying a constraint will be denoted as $\mathcal{S}_\infty(\lambda) \equiv \mathcal{S}(\lambda)$. Using the satisfaction sets it is then possible to define a partial ordering among the constraints, i.e., relate them to one another based on their difficulty to satisfy. A weakly-hard constraint is *harder to satisfy* if it is more

[6] In Paper IV we extend this notation to also encompass more appropriate languages in the real-time control systems setting.

restrictive in which sequences satisfy the constraint. Consider for instance the constraint $\lambda_1 = \binom{1}{1}$, which requires that every job meets its corresponding deadline. The constraint is extremely restrictive in what sequences it permits; in fact, the satisfaction set of this constraint only contains one sequence, $\mathcal{S}_N(\lambda_1) = \{1^N\}$. On the other hand, the constraint $\lambda_2 = \binom{0}{1}$ requires no job deadlines to be met to be satisfied; thus, all sequences satisfy this constraint, $\mathcal{S}_N(\lambda_2) = \Sigma^N$. Intuitively, since λ_1 is more restrictive than λ_2, we say that λ_1 *dominates* λ_2. We formalise this partial ordering in the following definition:

DEFINITION 4—CONSTRAINT DOMINANCE
Given two arbitrary weakly-hard constraints λ_1 and λ_2, we say that λ_1 dominates λ_2 (denoted $\lambda_1 \preceq \lambda_2$) if and only if all words satisfying λ_1 also satisfy λ_2. Formally,

$$\lambda_1 \preceq \lambda_2 \Leftrightarrow \mathcal{S}(\lambda_1) \subseteq \mathcal{S}(\lambda_2).$$

Definition 4 confirms that $\lambda_1 = \binom{1}{1}$ dominates $\lambda_2 = \binom{0}{1}$, because $\mathcal{S}(\lambda_1) \subseteq \mathcal{S}(\lambda_2)$. Many constraint dominance relations have been derived in literature [Bernat et al., 2001].[7] Additionally, the partial ordering motivates the notion of *constraint equivalence*

$$\lambda_1 \preceq \lambda_2 \wedge \lambda_2 \preceq \lambda_1 \Leftrightarrow \lambda_1 \equiv \lambda_2,$$

where $\wedge$ is the logical conjunction operator. The constraint equivalence is also expressible through the satisfaction sets, i.e., $\lambda_1 \equiv \lambda_2 \Leftrightarrow \mathcal{S}(\lambda_1) = \mathcal{S}(\lambda_2)$.

Despite not having gained a lot of traction in the research community, a real-time task can be subjected to *multiple* weakly-hard constraints. However, the nature of the weakly-hard constraints still require that every constraint is satisfied for a particular sequence. Since one of the main mathematical advantages of the weakly-hard constraints is that they are fully representable by the closed set that is their respective satisfaction sets; if a task τ satisfies a set of N constraints $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_N\}$, the joint satisfaction set has to be the intersection of all the individual satisfaction sets

$$\mathcal{S}(\Lambda) = \bigcap_{i=0}^{N} \mathcal{S}(\lambda_i).$$

The chosen task model is important when analysing the execution pattern of the real-time task, but implementation details, such as the scheduler's functionality, are typically not included in the analysis. In fact, it has been shown that the implementation's design choices significantly affect both the performance and safety properties of the system [Cervin, 2005]. This thesis addresses the discrepancy between the real-time models, control theoretical models, and the implementation specifications by including details about the implementation in the real-time control system analysis. For instance, consider the $k+3$-rd job in Figure 2.2; the behaviour of the

[7] In Paper III we extend the known orderings with two theorems relating the `AnyHit` and `RowHit` constraints, thus making it possible to relate *all* the different weakly-hard constraints.

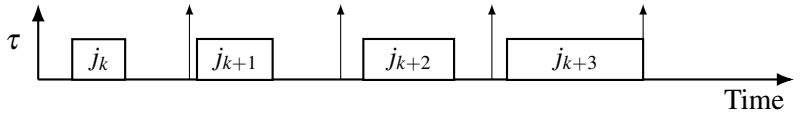


Figure 2.2 Example of the execution trace belonging to a task τ, where four jobs are depicted. The $k+3$-rd job reaches its corresponding deadline (the activation of the next job) without having completed its execution.

real-time system is undefined, regardless of the chosen task model. The function that job j_{k+3} is supposed to carry out remains unfinished, resulting in an unknown behaviour if the overrun deadline is left unmanaged. It is therefore crucial to include some details about the implementation in the system analysis.

In addition to orchestrating the tasks in the RTOS, the scheduler is also responsible for supervising the tasks overrunning their deadlines (denoted the *overrun handling strategy*). Different strategies have been developed to handle overruns in varying applications. In [Caccamo et al., 2002], the authors propose a method for avoiding deadline overruns by postponing the deadline of the job that requires more processor time to complete. An arbiter designed to drop certain jobs upon release (i.e., skipping them) is proposed in [Yoshimoto and Ushio, 2011]. However, as described in [Cervin, 2004], three of the simplest overrun handling strategies are:

- `Queue` – The naive approach involves letting the job overrunning its deadline to continue its execution whilst queueing up subsequent jobs. As soon as the executing job is finished, the first instance in the job queue is immediately released and activated. Instead of queuing all subsequent jobs, it is common to only queue the most recent job; this is typically denoted the `Queue(1)` strategy. However, the `Queue` strategies risk successive jobs being delayed enough to induce domino effects in the system.

- `Skip` – Under the `Skip` strategy (sometimes referred to as `Skip-Next` or `Continue`), the job overrunning its deadline is allowed to continue executing until completion. Unlike the `Queue` strategy, subsequent jobs are skipped (i.e., terminated before release) instead of being put into a job queue. Hence, the domino effects that can occur under `Queue` are avoided; this does however come at the cost of skipping a full job even in the presence of infinitesimal overruns.

- `Kill` – If a job overruns its deadline under the `Kill` strategy (sometimes referred to as `Abort`), the job is immediately terminated allowing the subsequent job to be released and activated on time. One of the main advantages with the `Kill` strategy comes from its binary outcome representation, i.e., either the job is completed or it is not. This fits well together with, for instance, the weakly-hard task models' language representation. On the other hand, a

drawback with `Kill` is that if the task function depends on an internal state, the part of the computation that was completed may need to be rolled back to a previous state via, e.g., memory checkpointing [Vogt et al., 2015]. Since such an operation requires additional overhead, it further increases the risk of missing the subsequent job's deadline. Additionally, if many consecutive job deadlines are overrun, the corresponding task risks never doing any actual work.

A framework for switching between `Kill` and `Skip` to drop delayed packets in arbitrated networked control systems is presented in [Soudbakhsh et al., 2018]. The authors of [Pazzaglia et al., 2018] discuss the performance of real-time control systems subject to the `AnyMiss` constraints with respect to both the `Kill` and `Skip` strategies; additionally, the authors discuss how the overrun handling strategy affect the freshness of the control signal, i.e., the age of the actuated control signal. In [Köhler and Ernst, 2019] the authors extend an existing method for computing weakly-hard guarantees in multi-component systems where deadline outcomes are considered binary events, i.e., adhering to the `Kill` strategy.

Separately from the scheduler's overrun strategy, it is important to analyse what happens when the actuators do not receive a fresh control command due to, for instance, a deadline overrun or a packet loss on the network. The actuators typically employ an *actuator mode* to decide which command to enact on the plant if no new control command has been received. The research community has proposed (among many other) smart actuator degradation policies [Ma et al., 2018] and adaptive compensation schemes [Xing et al., 2017] for actuators subject to intermittent faults. However, two of the simplest and most frequently employed policies are the `Zero` and `Hold` actuator modes [Schenato, 2009]. If no new control signal is received by the actuator, the `Hold` actuation mode holds the last received control command until a new directive is received. On the contrary, the `Zero` actuation mode stops acting on the plant if no new control command is received.[8] Choosing a proper actuator mode is non-trivial and is in general highly dependent on the plant and controller dynamics [Schenato, 2009]. For controllers with integrating dynamics, `Hold` is a good choice pressuming that the controller command is keeping the system close to its desired execution point. On the contrary, the `Zero` mode may be safer for plants with noisy or unstable dynamics, because avoiding to affect the system can be better than to act on outdated control commands.

2.1.1 Execution Modelling using State Machines

Ever since Liu and Layland proposed their simplistic task model [Liu and Layland, 1973], more expressive models have been sought to properly characterise the execution of real-time tasks. One of the more prominent methods to capture the task execution's expressiveness involved utilising directed acyclic graphs (DAGs) [Baruah,

[8] Paper I shows that the choice of actuation mode can significantly affect both stability and performance properties of the physical system, when the controller is subject to computational overruns.

2003; Chakraborty and Thiele, 2005; Stigge et al., 2011]. In fact, both graphs and *finite state machines* (FSM) are frequently used to monitor task execution and verify system safety [Kumar et al., 2012; Dai and Burns, 2020; Hertneck et al., 2020]

The use of finite state machines (also referred to as *finite state automata*) have moreover been applied to the research of deadline overrun modelling. In fact, for a task's jobs the computational overrun process have been modelled using finite state machines for both soft and firm real-time systems. In [Horssen et al., 2016] the authors model the `AnyMiss` weakly-hard constraint using an automaton where the transition events between states are represented by the job outcomes. A *Markov chain* (MC) model was used in [Ling and Lemmon, 2003] to model the dropout rate of sensor packets in a soft networked control system. [Kwak et al., 2001] utilise a Markov chain to select optimal time points for memory checkpointing in real-time systems where tasks may experience transient faults. Henceforth, we will sloppily refer to finite state machines as *automata*.[9]

Both automata and Markov chains are constructed from directed graphs, with the main difference that the automata transitions are deterministic by nature whilst the Markov chain requires a distribution of probabilities to model the transitions between states. In this thesis, both automata and MC are represented by an underlying graph $\mathcal{G} = (V, E)$, where V is the set of *vertices* (or *states*) and E is the set of *edges* (or *transitions*). However, the vertices and transitions symbolise different deadline overrun models for respectively the automata and Markov chain.

The automaton model is used to represent the feasible sequences of deadline hits and misses for the weakly-hard constraints.[10] Here, the underlying graph depend on the chosen constraint, i.e., $\mathcal{G}_\lambda = (V_\lambda, E_\lambda)$. Each vertex $v_i \in V_\lambda$ represents a word $w_i \in \mathcal{S}_k(\lambda)$, where k is the window length of λ. A transition $e_{i,j} = (v_i, v_j, c_{i,j}) \in E_\lambda$ is a triplet describing the transition condition $c_{i,j} \in \Sigma$ to get from vertex v_i to v_j. For instance, there exists a transition $e_{i,j} = (v_i, v_j, 0)$ between v_i and v_j if the sequence of deadline hits and misses w_i still satisfies the constraint λ if followed by a deadline miss (i.e., $c_{i,j} = 0$) and the resulting sequence can be represented by w_j.

When modelling stochastic computational overruns, the Markov chain models are naturally better-suited than the automaton model.[11] The MC is represented by a set of states $v_i \in V$ where the transitions between states is again a triplet $e_{i,j} = (v_i, v_j, p_{i,j}) \in E$. Unlike the automata model, the transition here defines a transitional probability $p_{i,j}$ between two states v_i and v_j, i.e., a transition from state v_i to state v_j occurs with probability $p_{i,j} \in [0, 1]$. Trivially, the cumulative probability of leaving any state v_i is 1

$$\sum_{v_j \in V} p_{i,j} = 1.$$

Note that $p_{i,i}$ is not necessarily 0, meaning that with probability $p_{i,i}$ state v_i remains

[9] To keep notation consistent with Papers III and IV.

[10] The automaton model is derived in Paper III and utilised in Paper IV.

[11] The Markov chain is utilised in Papers II and V.

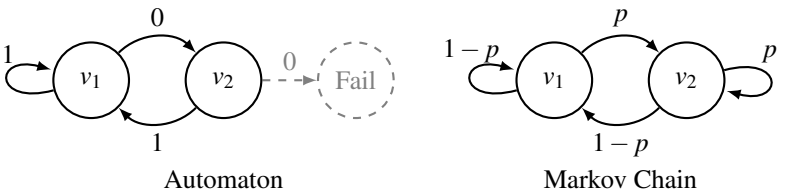


Figure 2.3 Graph representation example of an automaton (left) and Markov chain (right). Left: the automaton models the firm real-time constraint that at most one deadline overrun can exist in every window of two consecutive jobs. Right: the Markov chain models the soft real-time constraint in which the outcome of each job is an iid process with overrun probability p.

the active state. A Markov state is said to be *absorbing* if it, once entered, is never left, i.e., has probability $p_{i,i} = 1$. The transitional probability depends on the specific probability distribution.

To demonstrate the similarities and differences between the automaton and Markov chain, an example is shown in Figure 2.3. Here, the automaton is constructed from a weakly-hard real-time task subject to the constraint $\binom{x}{k} = \binom{1}{2}$, i.e., at least one job has to meet its deadline in every window of two consecutive job activations. Recall that the characters 0 and 1 respectively represent an overrun and a met deadline in the automaton's language. Then, the vertex v_1 represent a sequence with a met deadline as its most recent job outcome whilst v_2 correspond to a sequence where the last job overran its deadline. Since the constraint does not tolerate two consecutive deadline overruns, if a deadline overrun occur from vertex v_2 the constraint would be violated and the fail-state is entered. In comparison, the MC represent a soft real-time task where every job has an independent and identically distributed (iid) probability p of overrunning its deadline. Notice that the transitions in both the automaton and in the Markov chain represent the outcome of exactly one job execution. This is a deterministic transition (no probabilities involved) in the automaton, whilst it is stochastic for the MC.

2.2 Control Systems

By omitting the implementation details, such as the real-time system's hardware layer and operating system, the remaining components are the mathematical models of the plant and controllers. Figure 2.4 highlights how these models relate to the relevant system architecture. The mathematical models characterise the system's behaviour in time, typically via a dynamical system of differential (or difference)

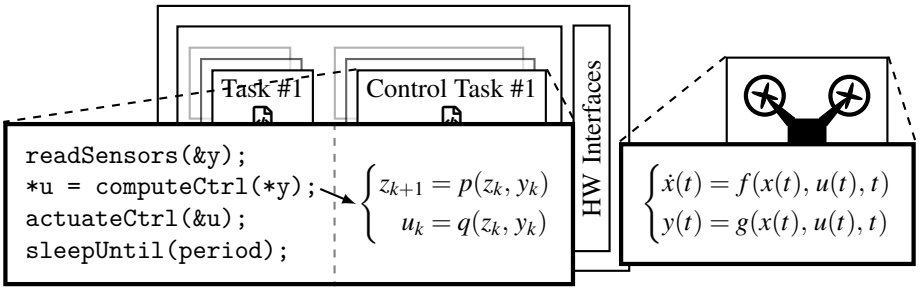


Figure 2.4 An example of the code snippet executed by the control task together with the mathematical representation of the control algorithm (left) and the mathematical model of the plant dynamics (right).

equations written on *state-space* form

$$\begin{cases} \dot{x}(t) = f(x(t), u(t), t) \\ y(t) = g(x(t), u(t), t). \end{cases} \tag{2.1}$$

Practically all real-world plants can be modelled using continuous-time, nonlinear state-space equations on the same form as Equation (2.1). The physical properties of the plant (e.g., velocity, position, rotation, etc.) are collected in the *state vector* $x(t) \in \mathbb{R}^{n_x}$, the exogenous signals affecting the plant are collected in the *input vector* $u(t) \in \mathbb{R}^{n_u}$, and the measurable quantities are collected in the *output vector* $y(t) \in \mathbb{R}^{n_y}$. The dynamical behaviour of the system is described by the functions f and g, which we say are *time-variant* if they explicitly depend on the variable t and *time-invariant* if they do not, i.e., if $f(x(t), u(t), t) = f(x(t), u(t))$ and $g(x(t), u(t), t) = g(x(t), u(t))$. A state-space system is linear if the functions f and g can be expressed as linear combinations of their arguments, i.e., a linear state-space system can be written as

$$\begin{cases} \dot{x}(t) = A_c(t)\,x(t) + B_c(t)\,u(t) \\ y(t) = C_c(t)\,x(t) + D_c(t)\,u(t). \end{cases} \tag{2.2}$$

The matrices $A_c(t) \in \mathbb{R}^{n_x \times n_x}$ and $B_c(t) \in \mathbb{R}^{n_x \times n_u}$ govern the state evolution of the plant while $C_c(t) \in \mathbb{R}^{n_y \times n_x}$ and $D_c(t) \in \mathbb{R}^{n_y \times n_u}$ describe its output process. The plant is said to be linear time-invariant (LTI) if none of the matrices in Equation (2.2) are time-dependent, e.g., $A_c(t) = A_c$.

If the plant is continuous, it is typically *discretised* to simplify the analysis, synthesis, and implementation of the controllers. Transforming the continuous-time plant model into its discrete-time equivalent is non-trivial and generally depend on how the continuous system is sampled, i.e., how the analog-to-digital converters

(ADC) and digital-to-analog converters (DAC) are designed. The DAC is located near the actuators and translates the digital value received from the controller to an analog actuator command. Similarly, the ADC transforms the analog sensor values to digital signals to be sent to the controlling hardware. One of the most commonly applied sampling techniques is the zero-order hold (ZOH) circuit, where the DAC holds the last converted analog signal constant until a new discrete value is received and converted. The sensors are generally configured to sample the plant at the discrete time instants when the control signal to the plant changes. In other words, if the DAC receives and converts discrete values at time instants $\{t_k\}_{k\in\mathbb{N}_>}$, the sensors sample the system in the same time points. For periodically sampled systems, the sampling instants are equidistant with a sampling period of T, i.e., $t_k = k \cdot T$. The resulting discrete-time, LTI model of the plant $\mathcal{P}$ is then

$$\mathcal{P} : \begin{cases} x_{k+1} = A\,x_k + B\,u_k + W\,w_k \\ y_k = C\,x_k + D\,u_k, \end{cases} \tag{2.3}$$

where the variable subscripts counts the discrete time samples since system startup, i.e., $x_k = x(k \cdot T)$ and $x_{k+1} = x(k \cdot T + T)$. Note that the process' stochastic disturbance process $w_k \in \mathbb{R}^{n_w}$ and dynamic matrix $W \in \mathbb{R}^{n_x \times n_w}$ have been added. The disturbance process w_k is typically a stochastic process with (assumed) known statistical properties, modelling the unknown plant dynamics and exogenous signals.

Similarly to how the plant is described using a dynamical system on state-space form, controllers are typically defined on the same format (denoted the controller's *control law*). Section 2.1 described how the control algorithm is implemented as a function governed by a task executing in the RTOS; modern controllers are hence unlikely to be continuous. Therefore, the discrete, time-invariant state-space representation of a general control law is

$$\begin{cases} z_{k+1} = p(z_k, y_k) \\ u_k = q(z_k, y_k). \end{cases} \tag{2.4}$$

Here, the *control state vector* $z_k \in \mathbb{R}^{n_z}$ represent the controller's dynamics, the *control signals* $u_k \in \mathbb{R}^{n_u}$ are inputs to the plant, and the functions p and q govern the dynamical behaviour of the controller. Intuitively, each time step k corresponds to one job activation of the control task. Similar to the plant, if p and q can be expressed as linear combinations of their arguments, i.e., $p(z_k, y_k) = F\,z_k + G\,y_k$ and $q(z_k, y_k) = H\,z_k + K\,y_k$, then the controller $\mathcal{C}$ is linear

$$\mathcal{C} : \begin{cases} z_{k+1} = F\,z_k + G\,y_k \\ u_k = H\,z_k + K\,y_k. \end{cases} \tag{2.5}$$

We say that a controller $\mathcal{C}$ is *stateless* (or *static*) if it has no dependence on the internal state z, i.e., $n_z = 0$. Furthermore, a *stateful* (or *dynamic*) controller $\mathcal{C}$ is any controller that depend on its internal state z, i.e., $n_z > 0$.

To improve timing predictability whilst also reducing input-output jitter one-step delay controllers are frequently employed.[12] The one-step delay is generally taken into consideration during the control design process, in other words, the control algorithm is designed for the control signal to be computed during the k-th control period and to be deployed by the actuators at the release of the $k+1$-st control job. In the real-time literature, one-step delay are commonly referred to as *logical execution time* (LET) controllers [Ernst et al., 2018; Gemlau et al., 2021; Kirsch and Sokolova, 2012]. The LET paradigm simplifies the schedulability analysis, improves timing predictability, and removes time-varying computational delays (e.g., release jitter, context switching overhead, and interrupts), but it comes at the cost of introducing a time delay in the actuation.

The vast treatise on synthesis and analysis of both linear and non-linear control laws that exists is likely too large to review in any thesis; however, some of the most known algorithms are outlined here. Arguably the most famous control structure is the proportional-integral-derivative (PID) controller, in which the error between the sensor measurements and the desired plant states are used to compute the new control signal [Åström and Hägglund, 2006]. The PID controller is linear and stateful, assuming that either the I or D part is included. From the domain of optimal control, the linear-quadratic regulator (LQR) was derived to analytically minimise a quadratic cost function penalising large control signals and plant state deviations. The linear and stateless LQR algorithm has been the subject to a large research effort [Goswami et al., 2012; Linsenmayer and Allgöwer, 2018], likely due to its elegant mathematical properties. Additionally, the LQR is a fundamental part of solving the linear-quadratic-Gaussian (LQG) problem [Åström and Wittenmark, 1997]. Like LQR, model predictive control (MPC) originate from optimal control [Allgöwer et al., 1999]; however, they differ in the cost function's structure. When LQR optimises the control structure over a full time window, MPC provides a receding time horizon solution while also supporting constraints on the trajectory, control state, and control signal.

Control theory is dedicated to developing models and algorithms (such as PID, LQR, LQG, and MPC) for designing p and q such that the control system's state follows a desired trajectory while minimising undesirable effects such as noise, disturbances, and large actuator commands. In Sections 2.2.1 and 2.2.2, the control law's objectives will be expanded upon, particularly discussing *stability* and *performance*. However, to discuss stability and performance in real-time control systems, it is crucial to first properly introduce *feedback*. The sensor signals that are received at the controller are processed by the control algorithm before the computed control signal is transmitted back to the actuators. Routing signals back into the plant like this is called *feedback control* (or *closed-loop control*). Typically, the closed-loop

[12] If the controller adopts a one-step delay approach, the order of the last two statements of the control algorithm in Figure 2.4 has to be swapped, suggesting that the second equation of the system of equations (2.5) instead should be read as $u_{k+1} = H z_k + K y_k$. However, the control law's reformulation is typically made implicitly by augmenting the controller's state with the control signal.

system (i.e., all components involved in the closed-loop control) is also described by a dynamical equation, acquired by combining Equations (2.3) and (2.5)

$$\tilde{x}_{k+1} = \Phi \tilde{x}_k + \Gamma_w w_k, \tag{2.6}$$

where $\tilde{x}_k$ is the closed-loop system's state vector, Φ is the closed-loop system matrix responsible for encoding the closed-loop system's dynamics, and Γ_w is the dynamic matrix of the exogenous stochastic variable w_k.

Thus far, the system execution has been assumed to be faultless. However, we are interested in analysing the real-time control system when the control tasks are subject to computational overruns. Despite only having system components that are discrete-time, linear, and time-invariant, the closed-loop system dynamics become time-variant when introducing computational faults. In place of Equation (2.5), the controller $\mathcal{C}$ is then characterised by

$$\mathcal{C} : \begin{cases} z_{k+1} = F_{\theta_k} z_k + G_{\theta_k} y_k \\ u_k = H_{\theta_k} z_k + K_{\theta_k} y_k, \end{cases} \tag{2.7}$$

where θ_k is the process governing the overruns, e.g., a weakly-hard sequence or Markov process. For instance, given a sequence w of job outcomes adhering to the weakly-hard `RowHit` constraint $\langle {}^{2}_{5} \rangle$, e.g., $w = 110011011$; indexing the k-th character in the word w as c_k, the outcome process is $\theta_k = c_k$. The controller dynamics then depend on whether the deadline was met (1) or overrun (0) as well as the scheduler's overrun policy, i.e., `Kill`, `Skip`, `Queue`, or some other deadline overrun strategy.

The changing control equations do in turn also change the closed-loop dynamics from Equation (2.6)

$$\tilde{x}_{k+1} = \Phi_{\theta_k} \tilde{x}_k + \Gamma_w w_k. \tag{2.8}$$

If θ_k is governed by a Markov process, then Equation (2.8) is denoted a *Markov jump linear system* (MJLS) [Feng et al., 1992]. Instead, if the process changing the closed-loop dynamics is deterministic, it is labeled a *switching system* [Liberzon, 2003]. We acknowledge that this simplified description of both Markov jump linear systems and switching systems is extremely coarse. In the upcoming sections, we will introduce notation for both MJLS and switching systems such that the stability and performance analyses are unambiguous.

2.2.1 Control System Stability

One of the fundamental objectives of all control synthesis is to *stabilise* the system. Stability theory is a broad domain and there exists many both weak and strong definitions of stability. For dynamical systems (such as control systems), a common notion of stability is that bounded perturbations in the initial conditions of a differential (or difference) equation results in bounded perturbations in the solution.[13] In

[13] Note that this implies that the disturbances do not affect the linear system's stability.

other words, if a system is *stable* for a bounded set of initial condition then the solution will not grow unbounded. Conversely, if a system is *unstable* the solution grows unbounded. As an example, if the drone system from Chapter 1 was unstable, under certain conditions the motor velocity would rapidly increase (or decrease) until the drone finally crashed (a highly undesirable behaviour).

In practice, these vague notions of stability are formalised in rigorous mathematical definitions. Two of the classic definitions of stability for *linear* dynamical systems are the Routh-Hurwitz stability criterion for continuous systems [Åstrom and Murray, 2008] and the Schur stability criterion for discrete systems [Åström and Wittenmark, 1997]. The criteria could be summarised by:

- A continuous-time linear system is said to be *Hurwitz stable* if all its characteristic polynomial's roots lie in the open left half-plane.
- A discrete-time linear system is said to be *Schur stable* if all its characteristic polynomial's roots lie inside the open unit disk.

In practice, these conditions directly correspond to verifying that all the system's eigenvalues are either strictly in the left half plane (continuous systems) or strictly inside the unit disc (discrete systems). The Hurwitz stability criterion for a continuous-time LTI system with system matrix Φ^c and the Schur stability criterion for a discrete-time LTI system with system matrix Φ is then

$$\begin{aligned} &\text{Hurwitz:} && \max\left\{\mathrm{Re}\left(\mathrm{eig}_i\left(\Phi^c\right)\right)\right\} < 0, \\ &\text{Schur:} && \max\left\{\left|\mathrm{eig}_i\left(\Phi\right)\right|\right\} < 1, \end{aligned}$$

where $\mathrm{eig}_i(\Phi)$ retrieves the i-th eigenvalue of Φ and $\mathrm{Re}(\cdot)$ extracts the real-part of a complex number (element-wise when applied to a vector). It is worth noting that the Schur stability criterion is dependent on the *spectral radius* of the system, i.e., the maximum asymptotic growth rate of the system. Denoting the spectral radius of a square matrix $\Phi \in \mathbb{R}^{n\times n}$ with $\rho(\Phi)$, it is then defined as[14]

$$\rho\left(\Phi\right) = \max\left\{\left|\mathrm{eig}_1\left(\Phi\right)\right|, \left|\mathrm{eig}_2\left(\Phi\right)\right|, \ldots, \left|\mathrm{eig}_n\left(\Phi\right)\right|\right\} = \lim_{k\to\infty} \|\Phi^k\|^{1/k}. \tag{2.9}$$

Deterministic Switching Stability The classical notions of linear systems stability do not hold when the closed-loop dynamics is time-variant. Instead, more powerful mathematical tools are needed, dependent on the definition of the switching process θ_k, i.e., whether the system is subject to *deterministic* or *stochastic* switching. By deterministic switching we mean systems where the switching process θ_k is a deterministic constraint, such as the weakly-hard constraints; in other words, a switching system where stability is independent of the probability of transitioning between different execution modes. On the other hand, stochastic switching refer to systems

[14] The second equality is a well-established relation known as Gelfand's formula. The equality holds true regardless of the chosen matrix norm.

where θ_k is governed by an entirely stochastic process, e.g., a Markov chain, and the switching stability is hence highly dependent on the transition probabilities. In this thesis, emphasis has been to analyse switching stability from the deterministic framework, using methods introduced next.

A useful tool for determining stability of most dynamical systems is *Lyapunov's second method* – more frequently referred to as *Lyapunov's stability criterion*. Essentially, the stability criterion is based on finding a function with particular properties (denoted the *Lyapunov function*) for the system under consideration; if a Lyapunov function exists, the system is globally asymptotically stable [Åström and Wittenmark, 1997]. The Lyapunov function is an energy function which is zero at the unique point of equilibrium, positive everywhere else, and decreases along all trajectories of the system. For instance, consider an oscillating pendulum; it will always move toward the position with the lowest energy, finally reaching its resting state when it is hanging freely downwards, i.e., where the potential energy of the pendulum has reached its equilibrium. Unfortunately, finding a suitable Lyapunov function is generally a very complex problem – in particular when the system experiences switching dynamics due to overruns.

Fortunately, the *joint spectral radius* (JSR) [Rota and Strang, 1960] has in recent years become the subject of intense research due to its role in (among others) discrete-time switching systems stability analysis. From its name, it might be clear that the joint spectral radius is a generalisation of the spectral radius from Equation (2.9). In fact, the JSR generalises the spectral radius to a set of matrices $\mathcal{A} = \{\Phi_1, \Phi_2, \ldots, \Phi_N\}$ (denoted the *switching set*), i.e., it represents the maximum asymptotic growth rate of the arbitrary switching between matrices in the set. For the switched dynamical system in Equation (2.8), each matrix $\Phi_i \in \mathcal{A}$ is a unique matrix describing one switching mode of θ_k and the JSR is then the maximal asymptotic growth rate of the closed-loop states $\tilde{x}_k$ under arbitrary switching of the matrices $\Phi_i \in \mathcal{A}$. Trivially, if the switching set only contains one matrix, i.e., $\mathcal{A} = \{\Phi\}$, then the joint spectral radius and the spectral radius are equivalent.

To simplify notation, we borrow the following definition from [Jungers, 2009]

$$\mathcal{A}^k \triangleq \{\Phi_1 \cdots \Phi_k \mid \Phi_i \in \mathcal{A}\}.$$

Thus, $\mathcal{A}^k$ is the set of *all* matrix multiplications involving k matrices from the set $\mathcal{A}$. Using the introduced notation, the following definition provides a formal definition of the JSR [Jungers, 2009].

DEFINITION 5—JOINT SPECTRAL RADIUS
Given a bounded set of matrices $\mathcal{A}$, the joint spectral radius (JSR) of $\mathcal{A}$ is defined as

$$\rho(\mathcal{A}) = \limsup_{k\to\infty} \left\{ \|\overline{\Phi}\|^{1/k} \mid \overline{\Phi} \in \mathcal{A}^k \right\}.$$

Since $\rho(\mathcal{A})$ correspond to the maximum asymptotic growth rate of the arbitrary switching between matrices in the set $\mathcal{A}$, if the worst-case sequence of matrices

$\overline{\Phi} \in \mathcal{A}^k$ has a bounded growth rate $\|\overline{\Phi}\| < 1$, it implies that the switching system $\tilde{x}_k = \overline{\Phi} \cdot \tilde{x}_0$ is bounded by the triangle inequality, i.e., $|\tilde{x}_k| \leq \|\overline{\Phi}\| \, |\tilde{x}_0| \to 0$. Stability of a switched dynamical system via the joint spectral radius is outlined in the following theorem (for more details and a formal proof we refer the interested reader to [Jungers, 2009]).

THEOREM 1—SWITCHING STABILITY
For any bounded set of matrices $\mathcal{A}$, the corresponding switched dynamical system is stable if and only if $\rho(\mathcal{A}) < 1$.

Furthermore, Theorem 1 raises an interesting question about the converse case: which is the switching sequence $\Phi_1 \cdots \Phi_k \in \mathcal{A}^k$ resulting in $\rho(\mathcal{A}) \geq 1$? To the best of this thesis' author's knowledge, there are currently no results outlining *which* sequence that destabilises the switched dynamical system.

The JSR is a powerful tool since it makes it possible to determine whether a switching system is asymptotically stable ($\rho(\mathcal{A}) < 1$) or not ($\rho(\mathcal{A}) \geq 1$). However, the problem of computing the exact JSR value, and thus also testing whether or not the system is stable, is in practice undecidable [Blondel and Tsitsiklis, 2000]. To overcome this problem, the research community have in the last decade set out to develop and improve approximation methods, bounding the JSR from above and below [Jungers, 2009]. Multiple different methods have been developed to approximate the JSR, such methods include (but are not limited to): branch and bound [Gripenberg, 1996], polytope [Protasov, 1996], ellipsoidal norms [Blondel et al., 2005; John, 2014], and sum-of-squares (SOS) relaxation methods [Parrilo and Jadbabaie, 2008; Wang et al., 2021b; Wang et al., 2021a].

An interesting relation to the Lyapunov stability is the fact that searching for a common ellipsoidal norm is equivalent to finding a *common quadratic Lyapunov function*. However, in [Ando and Shih, 1998] the authors describe a constructive method for generating asymptotically stable switching systems for which no common quadratic Lyapunov function exist. Instead, the authors of [Parrilo and Jadbabaie, 2008] prove that if the solution to the SOS problem in Equation (2.10) provides a pair $(p(x), \gamma)$ such that (i) the polynomial $p(x)$ is homogeneous and in the interior of the SOS cone and (ii) $\gamma < 1$, then $p(x)$ is positive definite and decreases along all trajectories, meaning that $p(x)$ is a Lyapunov function for the switching system defined by $\mathcal{A}$.

$$\begin{aligned} \rho_{\text{SOS},2d}(\mathcal{A}) = & \inf_{p(x) \in \mathbb{R}_{2d}[x], \gamma} \gamma \\ & \text{s.t.} \quad \begin{cases} p(x) & \text{is SOS} \\ \gamma^{2d} p(x) - p(\Phi_i x) & \text{is SOS, } \Phi_i \in \mathcal{A}. \end{cases} \end{aligned} \tag{2.10}$$

Here $\mathbb{R}_{2d}[x]$ is the set of homogeneous polynomials of degree $2d$. Additionally, the authors show how the upper bound on the JSR achieved by the SOS approach in Equation (2.10) is never weaker than the one obtained with a common quadratic Lyapunov function.

Trivially, since $\rho^{LB}(\mathcal{A}) \leq \rho(\mathcal{A}) \leq \rho^{UB}(\mathcal{A})$, if an approximation method provides an upper bound $\rho^{UB}(\mathcal{A})$ that is below one for the switched dynamical system, it is confirmed to be stable; conversely, if the lower bound $\rho^{LB}(\mathcal{A})$ is above one, the system is guaranteed unstable. The active research in this field continues to enhance both the bounds and algorithmic complexities, thus constantly improving the applicability of the JSR notion.

Stochastic Switching Stability If the overrun process in Equation (2.8) is governed by a Markov model, it is possible to utilise well-known results for Markov jump linear systems (MJLS) [Lincoln and Cervin, 2002; Feng et al., 1992; Nilsson et al., 1998]. While the switching system stability analysis (e.g., JSR) provides a deterministic stability certificate, i.e., the switching system stability analysis provides guarantees that the dynamical system will be stable under *all* configurations of the different modes, the MJLS stability analysis provides a stochastic certificate. This certificate does in turn provide probabilistic guarantees that the MJLS will *almost surely* converge to its equilibrium point [Feng et al., 1992].[15]

Since MJLS are stochastic by nature, the existing stability analyses typically investigate how the expected value of the system's states evolve in time. Arguably the most well-known stability analysis methods are *stochastic stability* (SS) and different variants of *mean-square stability* (MSS) [Feng et al., 1992]. Paraphrased from [Costa et al., 2005], we say that the dynamical system in (2.8), governed by a Markov process θ_k, is mean-square stable if

$$\lim_{k\to\infty} \mathbb{E}\left[\|\tilde{x}_k\|^2\right] < \infty.$$

In practice, testing for mean-square stability implies verifying that the spectral radius of the operator Ψ, i.e., the operator defining the Markov jump linear system's covariance, is less than one. We denote an N-state Markov chain's transition probability matrix with $\Pi = \{p_{i,j}\} \in \mathbb{R}^{N\times N}$; this matrix specifies the probability of transitioning from the i-th to the j-th Markov state. Additionally, I_n is defined as the size $n \times n$ identity matrix and $\otimes$ denotes the Kronecker product. The following theorem then holds [Costa et al., 2005].

THEOREM 2—MEAN-SQUARE STABILITY
The dynamical system in Equation (2.8)*, governed by an N-state Markov chain with transition probability matrix* $\Pi \in \mathbb{R}^{N\times N}$*, is MSS if*

$$\rho(\Psi) < 1,$$

where

$$\Psi = \left(\Pi^{\mathrm{T}} \otimes I_{n^2}\right) \cdot \mathrm{blkdiag}\left(\left\{\Phi_i^{\mathrm{T}} \otimes \Phi_i\right\}_{i\in\{1,2,\ldots,N\}}\right).$$

[15] In probability theory, the notion of *almost surely* means that an event will happen with probability 1.

In practice, Theorem 2 simplifies the MSS testing and implies almost sure stability of the expected dynamics. Finding the spectral radius of a matrix is intuitively much faster than, for instance, the joint spectral radius (which has already been discussed to be undecidable). This is one of the main advantages with the stochastic certificate received by the MSS in comparison to the deterministic one received by computing the JSR.

2.2.2 Control System Performance

Stability analysis, such as the one discussed in Section 2.2.1 is undeniably a black and white concept. A control system can for instance neither be 90% nor 10% stable; the system is either stable or it is not. This stands in contrast to the notion of *control system performance* which is a more ambivalent and fluctuating concept. Performance as a metric to be measured exists in many differing domains, however, we will unequivocally use the notion of *performance* in place of *control system performance*, unless otherwise stated.

Evaluating the performance of a control system involves first defining what is important for the application in focus. As mentioned in Section 2.2, the objective of control synthesis is to design the functions p and q from Equation (2.4) such that some predetermined requirements are met. Aside from the obvious stability necessity, the requirements typically involve (but are not limited to): (i) accurately following a desired system trajectory, (ii) quick convergence to said trajectory, (iii) efficiently rejecting process disturbances, (iv) minimising the control effort without significantly affecting other objectives, and (v) estimating or predicting the system state by eliminating inaccuracies and measurement errors. How the different control objectives are valued is dependent on the specific application. Ideally, any controller's ultimate goal is to satisfy these objectives perfectly. Unfortunately, there exists a trade-off between the controller's *robustness* and *tracking performance*, i.e., how well it handles respectively disturbances and setpoint changes. The trade-off expresses itself implicitly as a limit on how efficiently the desired setpoint can be tracked if disturbances and unmodeled dynamics are also to be effectively rejected.

To judge how well the synthesised controller performs, a *performance analysis* is frequently employed. The analysis method uses a *performance metric* to qualitatively measure the performance of the controller, i.e., the performance metric produces a descriptive value for the specific controller. Using the performance measure, it is then possible to compare the quality of one controller to another. The analysis is typically simulation-based, meaning that the performance is evaluated on a virtual replica (or model) of the system. Intuitively, performance analysis methods provide a coarse estimation of how well the controller will perform when connected to the actual system. To improve the results, it is thus necessary to run the controller also on the physical hardware. However, this can become costly in case there are undetected problems with either the control or real-time system design. The system is therefore first evaluated in extensive simulations and tests [Mandrioli, 2022].

Innumerable different methods for analysing control system performance have been developed. This thesis facilitates a performance analysis of real-time control systems subject to computational overruns using a stochastic quadratic cost function

$$J_k = \mathbb{E}\left[\tilde{x}_k^{\mathrm{T}} Q \tilde{x}_k\right]. \tag{2.11}$$

Here, J_k is the closed-loop state's cost increment in the k-th period and Q is a positive semidefinite weight matrix penalising large closed-loop state deviations. The model is based on well-known theory for linear stochastic systems [Åström, 1970; Åström and Wittenmark, 1997; Nilsson et al., 1998; Cervin et al., 2019].

We assume that the desired reference point for the closed-loop state $\tilde{x}_k$ is zero and that the disturbance w_k is an independent, discrete-time, zero mean, Gaussian white noise process with covariance $\mathbb{E}\left[w_k w_k^{\mathrm{T}}\right] = R_1$. The assumptions may seem overly restrictive, but we emphasise that more elaborate and advanced disturbance processes (and reference trajectories) can be realised by augmenting the closed-loop system with additional states describing the exogenous dynamics.[16] With these assumptions, together with the initial condition on the closed-loop state's mean value and covariance being respectively $\mathbb{E}[\tilde{x}_0] = m_0$ and $\operatorname{cov}(\tilde{x}_0) = R_0$, it is possible to describe how to evaluate Equation (2.11).

The expression in Equation (2.11) is difficult to compute in practice. Instead, it is easier to first transform the cost J_k into an equivalent form before evaluating it. We define the closed-loop system's mean value and covariance functions as m_k and P_k respectively

$$m_k = \mathbb{E}[\tilde{x}_k], \quad P_k = \operatorname{cov}(\tilde{x}_k) = \mathbb{E}\left[\tilde{x}_k \tilde{x}_k^{\mathrm{T}}\right] - \mathbb{E}[\tilde{x}_k]\,\mathbb{E}[\tilde{x}_k]^{\mathrm{T}}. \tag{2.12}$$

Note that because $\mathbb{E}[w_k] = 0$, the mean value of the state adhere to the following dynamical equation

$$m_{k+1} = \mathbb{E}[\Phi \tilde{x}_k + \Gamma_w w_k] = \Phi m_k.$$

From (2.12) it then follows that (2.11) can equivalently be written as [Bates, 2011]

$$\begin{aligned} J_k &= \mathbb{E}\left[\tilde{x}_k^{\mathrm{T}} Q \tilde{x}_k\right] \\ &= \operatorname{tr}\left(\mathbb{E}\left[\tilde{x}_k^{\mathrm{T}} Q \tilde{x}_k\right]\right) \\ &= \operatorname{tr}\left(Q\,\mathbb{E}\left[\tilde{x}_k \tilde{x}_k^{\mathrm{T}}\right]\right) \\ &= \operatorname{tr}\left(Q\left(P_k + m_k m_k^{\mathrm{T}}\right)\right) \\ &= \operatorname{tr}(Q P_k) + m_k^{\mathrm{T}} Q m_k. \end{aligned} \tag{2.13}$$

Here, $\operatorname{tr}(\cdot)$ computes the trace of its argument. If the condition $\mathbb{E}[\tilde{x}_0] = m_0 = 0$ is enforced, the expression can be reduced to solving $J_k = \operatorname{tr}(Q P_k)$.

[16] In both Paper I and II we augment the system state to model respectively brown noise (integrated white noise) and reference trajectories.

Solving Equation (2.13) still requires knowledge about how the closed-loop state's covariance P_k propagates in time. Recalling that $\tilde{x}_k$ and w_k are independent, evolving the covariance results in the following dynamical equation

$$\begin{aligned} P_{k+1} &= \operatorname{cov}(\tilde{x}_{k+1}) \\ &= \mathbb{E}\left[\tilde{x}_{k+1}\tilde{x}_{k+1}^{\mathrm{T}}\right] - \mathbb{E}\left[\tilde{x}_{k+1}\right]\mathbb{E}\left[\tilde{x}_{k+1}\right]^{\mathrm{T}} \\ &= \mathbb{E}\left[\left(\Phi\tilde{x}_k + \Gamma_w w_k\right)\left(\Phi\tilde{x}_k + \Gamma_w w_k\right)^{\mathrm{T}}\right] - \Phi m_k m_k^{\mathrm{T}}\Phi^{\mathrm{T}} \\ &= \Phi\mathbb{E}\left[\tilde{x}_k\tilde{x}_k^{\mathrm{T}}\right]\Phi^{\mathrm{T}} + \Gamma_w\mathbb{E}\left[w_k w_k^{\mathrm{T}}\right]\Gamma_w^{\mathrm{T}} - \Phi m_k m_k^{\mathrm{T}}\Phi^{\mathrm{T}} \\ &= \Phi\left(\operatorname{cov}(\tilde{x}_k) + m_k m_k^{\mathrm{T}}\right)\Phi^{\mathrm{T}} + \Gamma_w\mathbb{E}\left[w_k w_k^{\mathrm{T}}\right]\Gamma_w^{\mathrm{T}} - \Phi m_k m_k^{\mathrm{T}}\Phi^{\mathrm{T}} \\ &= \Phi P_k\Phi^{\mathrm{T}} + \Gamma_w R_1\Gamma_w^{\mathrm{T}}. \end{aligned} \tag{2.14}$$

Equations (2.13) and (2.14) establish the foundation for the performance analysis on a nominally executing dynamical system, such as the one in (2.6). When the control task starts experiencing deadline overruns, the switching dynamics alters the performance analysis accordingly, i.e., switching Φ for Φ_{θ_k} in Equation (2.14).[17]

The research community developed multiple different tools to simplify the performance analysis of real-time embedded (or networked) control systems [Ohlin et al., 2006]. For instance, `JitterBug` [Lincoln and Cervin, 2002] and `JitterTime` [Cervin et al., 2019] are tools to precisely evaluate the quadratic cost function considered in this thesis (2.13) for mixed continuous-/discrete-time system components.

[17] In Papers I and II we make additional alterations to the performance analysis to properly facilitate it under the specific system conditions.

3

Contribution

The main scientific contribution of this thesis is presented in the five papers that follow this chapter. This chapter provides an overview of the scientific contributions of each of the five publications to the real-time control research. Additionally, each author's contribution to the papers is described in detail. We conclude the chapter with Section 3.2, which lists additional peer-reviewed publications that were excluded from this thesis. These publications have been omitted to improve the cohesion of this manuscript, as they cover tangential topics with respect to the research presented in the thesis.

Difference Between Published and Included Version There are two differences between the published papers and the version included in this thesis: (i) the language and mathematical notation of the papers presented in this thesis have been unified for readability and consistency, (ii) figures and tables have been resized to match the format of the thesis.

3.1 Included Papers

Paper I

N. Vreman, A. Cervin, and M. Maggio (2021). "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.15

Scientific Summary In Paper I, linear control system stability and performance is analysed when the controller experiences persistent computational blackouts. Historically, the models to analyse such events have been either stochastic or overly conservative. These models have been thoroughly investigated in the literature, but few have been shown to achieve similar results on industrial applications.

On the contrary, Paper I proposes a new fault model that handles consecutive computational overruns whilst also encapsulating graceful recovery back to nominal execution conditions. The fault model is coupled together with different actuator (`Zero` and `Hold`) and scheduling strategies (`Kill` and `Skip`) to provide a holistic system model. To properly analyse the specific fault model, a methodology for analysing the stability and performance is also derived. The stability analysis is based on a switched system stability approach, whilst the performance analysis is based on evolving a quadratic cost function in time.

To reinforce the analysis methodology's applicability to industrial applications, an experimental campaign is carried out. The paper analyses both a non-linear physical system (a Furuta pendulum) and a set of linear system models representative of the process industry. From the analysis performed on the physical system, the paper shows that it is possible to draw accurate conclusions about the physical process' behaviour using simulation results.

Contribution Statement The idea of analysing control system stability using fault models that are matching industrial needs was proposed by M. Maggio. The physical testing environment and the experiments were implemented and executed by N. Vreman, who also developed the fault model and its formalisation. The switching stability analysis was derived by M. Maggio and N. Vreman. The method for analysing the system performance was developed together by A. Cervin and N. Vreman. The manuscript writing and editing effort was shared among all three authors.

Paper II

N. Vreman, C. Mandrioli, and A. Cervin (2022a). "Deadline-Miss-Adaptive Controller Implementation for Real-Time Control Systems". In: *IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. DOI: 10.1109/RTAS54340.2022.00010

Scientific Summary Many different robust controllers exists, specifically targeting control systems where the controller can experience timing faults. The main advantage of using these synthesis methods is that they can guarantee a priori system stability, under their respective fault model. Paper II identifies some limitations of previous work, namely: (i) complex design methods, (ii) poor performance in nominal conditions, and (iii) strong assumptions on the control law's structure.

To address the shortcomings of previous synthesis methods, the paper proposes an adaptive control law that: (i) is simple to design and implement, (ii) requires minimal system knowledge, and (iii) does not affect the nominal control performance. The adaptive controller assumes that there exists a controller that has been synthesised with respect to a specific control system's specification under nominal conditions. Assuming that the `Kill` strategy is employed, the adaptation then alters the controller matrices (based on the number of computational overruns) to quickly restore nominal control performance. To analyse the performance gain of the adap-

tive controller, the existing stochastic performance analyses based on Markov jump linear system's theory is extended to compute the *relative performance degradation*, i.e., the performance degradation caused by using a controller other than the ideal controller.

The adaptive controller is validated on a physical process (a Ball and Beam) and on the set of process industrial models, used also in Paper I. Significant performance gain is shown despite having an immense number of computational overruns. Additionally, since the proposed method does not guarantee any a priori stability guarantees, the paper analyses the switched system stability a posteriori.

Contribution Statement The initial observation, that many fault-tolerant controllers are restricted to overly specific system conditions, was made by C. Mandrioli. A set of limitations was identified together by C. Mandrioli and N. Vreman; who also derived the adaptive controller addressing these limitations. A. Cervin derived the performance analysis to validate the controller. The adaptive controller was implemented on the Ball and Beam plant by N. Vreman, and the data was collected by both N. Vreman and C. Mandrioli. The manuscript writing and editing effort was shared among all three authors.

Paper III

N. Vreman, R. Pates, and M. Maggio (2022b). "WeaklyHard.jl: Scalable Analysis of Weakly-Hard Constraints". In: *IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. DOI: 10.1109/RTAS54340.2022.00026

Scientific Summary Ever since their introduction, the weakly-hard models have been steadily increasing in both academic and industrial popularity. The existing tools and research have focused on single constraints, in particular the `AnyMiss` constraint. In certain domains, e.g., control systems, this choice is likely motivated by the `AnyMiss` constraints popularity rather than its fitness to the problem statement.

Paper III aspires to (i) change the focus from the `AnyMiss` constraint to the `RowHit` constraint and (ii) propose a scalable, open-source tool (`WeaklyHard.jl`) that can be used to analyse *all* the weakly-hard constraints. In order to switch the attention to the `RowHit` constraint, we provide two novel theorems on the relation between it and the `AnyHit` constraint, finalising the relationship graph between all the weakly-hard constraints. The `WeaklyHard.jl` tool employs an automaton model to describe the discrete-time execution of a task subject to a specific weakly-hard constraint. Additionally, `WeaklyHard.jl` is the first tool to address *sets* of weakly-hard constraints. The paper reports the results of an experimental campaign that addresses both scalability of `WeaklyHard.jl` compared to the state-of-the-art alternative and tests the novel features of `WeaklyHard.jl`.

Contribution Statement The idea behind the paper and the tool was developed by N. Vreman. The tool and its underlying algorithms was developed by N. Vreman, with input from M. Maggio. The formulation of the theorems relating the `AnyHit` and `RowHit` constraints were derived by N. Vreman, with proof-sketches that R. Pates helped refining. The manuscript was written by N. Vreman and M. Maggio, with input from R. Pates.

Paper IV

N. Vreman, P. Pazzaglia, V. Magron, J. Wang, and M. Maggio (2022c). "Stability of Linear Systems Under Extended Weakly-Hard Constraints". *IEEE Control Systems Letters* **6**. DOI: 10.1109/LCSYS.2022.3179960

Scientific Summary Despite the weakly-hard models having steadily gained more traction in the analysis of control systems, the weakly-hard model fails at expressing some characteristics of the implementation of control systems, e.g., how the actual controller implementation deals with deadline misses. Paper IV rectifies this discrepancy, by extending the formal language defining the weakly-hard constraints, and connecting the analysis on weakly-hard control systems to the actual implementation of controllers.

The paper uses the automata-based model produced by `WeaklyHard.jl` (presented in Paper III). A post-processing step is proposed that refines the automaton and handles the analysis of *extended weakly-hard constraints*, the extension being the implementation's characteristics. The paper presents a stability analysis based on switching linear systems, that advances the state of the art of analysing implementation of control systems.

The dynamics of the linear control system is Kronecker lifted with the adjacency matrix of the automaton to obtain an equivalent system model, now also subject to the extended weakly-hard constraint. This system model can then be analysed using any method applicable to arbitrary switching systems. The paper uses a JSR approach, but other alternatives can be envisioned. In addition to the stability analysis, the paper discusses theoretical results relating the real-time constraints with the control system stability.

Contribution Statement P. Pazzaglia came up with the idea for the paper after discussions with N. Vreman. The theoretical results presented in the paper were developed together by N. Vreman and P. Pazzaglia. N. Vreman extended the functionality of `WeaklyHard.jl` with the ability to handle a more expressive formal language for implementation concerns. V. Magron and M. Maggio discussed the JSR analysis, which was implemented by J. Wang. The manuscript was written by P. Pazzaglia and N. Vreman, with input and revisions from co-authors.

Paper V

N. Vreman and M. Maggio (2023). "Stochastic Stability Analysis of Control Systems Subject to Communication and Computation Faults". *Submitted to ACM SIGBED International Conference on Embedded Software (EMSOFT)*

Scientific Summary Research into fault-tolerant real-time control systems has been split between focusing on networked systems experiencing communication dropouts, i.e., packet losses, or embedded systems suffering computational overruns. Despite carrying very different information, the computational overruns have been compared to communication dropouts when analysing the system dynamics, and vice versa. The paper aims to clarify this discrepancy by analysing what happens when these faults can occur simultaneously, as is the case for real implementations.

Most modern results on switched system analysis provide certificates of stability or performance under worst-case conditions. However, the worst-case conditions may be exceedingly rare, resulting in overly conservative guarantees. Instead, typical methods for deriving bounds on both packet losses and computational overruns are probabilistic by nature. By modelling both the packet losses on the input/output channels (to and from the plant) and the computational overruns as stochastic variables, we are able to utilise powerful results from Markov Jump Linear Systems theory, providing stability contracts with *almost sure* convergence guarantees.

We apply the analysis to two case studies from the recent literature and show their robustness to a comprehensive set of faults. The analysis clearly shows how the systems are distinctly affected by either individually or jointly occurring computational and communication faults.

Contribution Statement M. Maggio proposed the idea of looking into the analysis of systems with a combination of faults. The implementation of the analysis and the generation of the results was done by N. Vreman, with input from M. Maggio. The manuscript writing was shared between the two authors with N. Vreman taking a leading role.

3.2 Additional Publications

The author of this thesis has also contributed to the following peer-reviewed publications.

M. Gunnarsson, N. Vreman, and M. Maggio (2023). "Trusted Execution of Periodic Tasks for Embedded Systems". In: *IFAC World Congress*

M. Nyberg Carlsson, N. Vreman, and A. Cervin (2023). “Timing-Robust Control over the Cloud Using On-Line Parametric Optimization”. In: *IFAC World Congress*

K. Krüger, N. Vreman, R. Pates, M. Maggio, M. Völp, and G. Fohler (2021). “Randomization as Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Real-Time Systems with Task Replication”. *Leibniz Transactions on Embedded Systems* **7**:1. DOI: 10.4230/LITES.7.1.1

N. Vreman and C. Mandrioli (2020). “Evaluation of Burst Failure Robustness of Control Systems in the Fog”. In: *2nd Workshop on Fog Computing and the IoT (Fog-IoT)*. Vol. 80. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-95977-144-3. DOI: 10.4230/OASIcs.Fog-IoT.2020.8

N. Vreman, R. Pates, K. Krüger, G. Fohler, and M. Maggio (2019). “Minimizing Side-Channel Attack Vulnerability via Schedule Randomization”. In: *IEEE 58th Conference on Decision and Control (CDC)*. DOI: 10.1109/CDC40024.2019.9030144

N. Vreman and M. Maggio (2019). “Multilayer Distributed Control over 5G Networks: Challenges and Security Threats”. In: *Proceedings of the Workshop on Fog Computing and the IoT (Fog-IoT)*. IoT-Fog ’19. Association for Computing Machinery, New York, NY, USA. ISBN: 9781450366984. DOI: 10.1145/3313150.3313223

Bibliography

Abeni, L., D. Fontanelli, L. Palopoli, and B. Villalba Frías (2017). "A markovian model for the computation time of real-time applications". In: *International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–6. DOI: 10.1109/I2MTC.2017.7969878.

Åkesson, B., M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis (2020). "An empirical survey-based study into industry practice in real-time systems". In: *41st IEEE Real-Time Systems Symposium (RTSS)*.

Allgöwer, F., T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright (1999). "Nonlinear predictive control and moving horizon estimation — an introductory overview". In: Frank, P. M. (Ed.). *Advances in Control*. Springer London, London, pp. 391–449. ISBN: 978-1-4471-0853-5.

Ando, T. and M.-h. Shih (1998). "Simultaneous contractibility". *SIAM Journal on Matrix Analysis and Applications* **19**:2, pp. 487–498. DOI: 10.1137/S0895479897318812.

Åstrom, K. J. and R. M. Murray (2008). *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, USA. ISBN: 0691135762.

Åström, K. J. and B. Wittenmark (1997). *Computer-Controlled Systems (3rd Ed.)* Prentice-Hall, Inc., USA. ISBN: 0133148998.

Åström, K. (1970). *Introduction to stochastic control theory*. Vol. 70. Mathematics in science and engineering. Academic Press, United States. ISBN: 0-12-065650-7.

Åström, K. and T. Hägglund (2006). *Advanced PID Control*. English. ISA - The Instrumentation, Systems and Automation Society. ISBN: 978-1-55617-942-6.

Baruah, S. K. (2003). "Dynamic- and static-priority scheduling of recurring real-time tasks". *Real-Time Systems* **24**:1, pp. 93–128. ISSN: 0922-6443. DOI: 10.1023/A:1021711220939.

Bates, D. (2011). *Quadratic forms of random variables*. URL: https://pages.stat.wisc.edu/~st849-1/lectures/Ch02.pdf (visited on 2023-03-02).

Bernat, G., A. Burns, and A. Liamosi (2001). "Weakly hard real-time systems". *IEEE Transactions on Computers* **50**:4, pp. 308–321. DOI: 10.1109/12.919277.

Blondel, V. D., Y. Nesterov, and J. Theys (2005). "On the accuracy of the ellipsoid norm approximation of the joint spectral radius". *Linear Algebra and its Applications* **394**, pp. 91–107. ISSN: 0024-3795. DOI: 10.1016/j.laa.2004.06.024.

Blondel, V. D. and J. N. Tsitsiklis (2000). "The boundedness of all products of a pair of matrices is undecidable". *Systems & Control Letters* **41**:2, pp. 135–140. ISSN: 0167-6911. DOI: 10.1016/S0167-6911(00)00049-9.

Bradley, K., S. Cheung, N. Puketza, B. Mukherjee, and R. Olsson (1998). "Detecting disruptive routers: a distributed network monitoring approach". *IEEE Network* **12**:5, pp. 50–60. DOI: 10.1109/65.730751.

Brandenburg, B. (2011). *Scheduling and locking in multiprocessor real-time operating systems*. PhD thesis, p. 614. ISBN: 978-1-267-25618-8.

Caccamo, M., G. Buttazzo, and L. Sha (2002). "Handling execution overruns in hard real-time control systems". *IEEE Transactions on Computers* **51**:7, pp. 835–849. DOI: 10.1109/TC.2002.1017703.

Cervin, A., P. Pazzaglia, M. Barzegaran, and R. Mahfouzi (2019). "Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems". In: *IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1025–1032.

Cervin, A. (2004). *Merging Real-Time and Control Theory for Improving the Performance of Embedded Control Systems*. Department of Computer Engineering and Systems Science, University of Pavia, Italy.

Cervin, A. (2005). "Analysis of overrun strategies in periodic control tasks". *IFAC Proceedings Volumes* **38**:1. 16th IFAC World Congress, pp. 219–224. ISSN: 1474-6670. DOI: 10.3182/20050703-6-CZ-1902.01076.

Cervin, A., B. Lincoln, J. Eker, K.-E. Årzén, and G. Buttazzo (2004). "The jitter margin and its application in the design of real-time control systems". In: *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*.

Chakraborty, S. and L. Thiele (2005). "A new task model for streaming applications and its schedulability analysis". In: *Design, Automation and Test in Europe*, 486–491 Vol. 1. DOI: 10.1109/DATE.2005.26.

Comer, D. (2013). *Internetworking with TCP/IP*. 6th ed. Pearson. ISBN: 013608530X.

Costa, O. L. V., R. P. Marques, and M. D. Fragoso (2005). *Discrete-Time Markov Jump Linear Systems*. Probability and Its Applications. Springer London. ISBN: 978-1-85233-761-2.

Dai, X. and A. Burns (2020). "Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems". *Journal of Systems Architecture* **103**, p. 101691. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2019.101691.

Drepper, U. and I. Molnar (2003). "The native posix thread library for linux". *White Paper, Red Hat Inc* **10**:2, pp. 22–42.

Ernst, R., S. Kuntz, S. Quinton, and M. Simons (2018). "The logical execution time paradigm: new perspectives for multicore systems". *Dagstuhl Reports* **8**, pp. 122–149.

Feng, X., K. Loparo, Y. Ji, and H. Chizeck (1992). "Stochastic stability properties of jump linear systems". *IEEE Transactions on Automatic Control* **37**:1, pp. 38–53. DOI: 10.1109/9.109637.

Friebe, A. (2022). *Timing and schedulability analysis of real-time systems using hidden markov models*.

Gemlau, K.-B., L. Köhler, R. Ernst, and S. Quinton (2021). "System-level logical execution time: augmenting the logical execution time paradigm for distributed real-time automotive software". *ACM Transactions on Cyber-Physical Systems* **5**:2. ISSN: 2378-962X. DOI: 10.1145/3381847.

Goswami, D., M. Lukasiewycz, R. Schneider, and S. Chakraborty (2012). "Time-triggered implementations of mixed-criticality automotive software". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1227–1232. DOI: 10.1109/DATE.2012.6176680.

Gracioli, G., A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni (2015). "A survey on cache management mechanisms for real-time embedded systems". *ACM Computing Surveys* **48**:2. ISSN: 0360-0300. DOI: 10.1145/2830555.

Gripenberg, G. (1996). "Computing the joint spectral radius". *Linear Algebra and its Applications* **234**, pp. 43–60. ISSN: 0024-3795. DOI: 10.1016/0024-3795(94)00082-4.

Gunnarsson, M., N. Vreman, and M. Maggio (2023). "Trusted Execution of Periodic Tasks for Embedded Systems". In: *IFAC World Congress*.

Hansman, S. and R. Hunt (2005). "A taxonomy of network and computer attacks". *Computers & Security* **24**:1, pp. 31–43. ISSN: 0167-4048. DOI: 10.1016/j.cose.2004.06.011.

Hertneck, M., S. Linsenmayer, and F. Allgöwer (2020). "Stability analysis for nonlinear weakly hard real-time control systems". *IFAC-PapersOnLine* **53**:2. 21st IFAC World Congress, pp. 2594–2599. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.307.

Hopcroft, J., R. Motwani, and J. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA. ISBN: 0321455363.

Horssen, E. P. van, A. R. B. Behrouzian, D. Goswami, D. Antunes, T. Basten, and W. P. M. H. Heemels (2016). "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses". In: *2016 European Control Conference (ECC)*, pp. 2571–2577. DOI: 10.1109/ECC.2016.7810677.

John, F. (2014). "Extremum problems with inequalities as subsidiary conditions". In: Giorgi, G. et al. (Eds.). *Traces and Emergence of Nonlinear Programming*. Springer Basel, pp. 197–215. ISBN: 978-3-0348-0439-4. DOI: 10.1007/978-3-0348-0439-4_9.

Jungers, R. (2009). *The Joint Spectral Radius: Theory and Applications*. Lecture Notes in Control and Information Sciences. Springer Berlin, Heidelberg. ISBN: 9783540959793.

Karray, F., M. W. Jmal, A. Garcia-Ortiz, M. Abid, and A. M. Obeid (2018). "A comprehensive survey on wireless sensor node hardware platforms". *Computer Networks* **144**, pp. 89–110. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2018.05.010.

Kirsch, C. and A. Sokolova (2012). "The logical execution time paradigm". In: *Advances in Real-Time Systems*. Springer Berlin Heidelberg, pp. 103–120. ISBN: 978-3-642-24349-3.

Köhler, L. and R. Ernst (2019). "Improving a compositional timing analysis framework for weakly-hard real-time systems". In: *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 228–240. DOI: 10.1109/RTAS.2019.00027.

Krüger, K., N. Vreman, R. Pates, M. Maggio, M. Völp, and G. Fohler (2021). "Randomization as Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Real-Time Systems with Task Replication". *Leibniz Transactions on Embedded Systems* **7**:1. DOI: 10.4230/LITES.7.1.1.

Kumar, P., D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele (2012). "A hybrid approach to cyber-physical systems verification". In: *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. Association for Computing Machinery, New York, NY, USA, pp. 688–696. ISBN: 9781450311991. DOI: 10.1145/2228360.2228484.

Kwak, S. W., B. J. Choi, and B. K. Kim (2001). "An optimal checkpointing-strategy for real-time control systems under transient faults". *IEEE Transactions on Reliability* **50**:3, pp. 293–301. DOI: 10.1109/24.974127.

Liberzon, D. (2003). *Switching in Systems and Control*. Systems & control. Birkhauser. ISBN: 9783764342975.

Lincoln, B. and A. Cervin (2002). "Jitterbug: a tool for analysis of real-time control performance". In: *IEEE Conference on Decision and Control*. Vol. 2, pp. 1319–1324. DOI: 10.1109/CDC.2002.1184698.

Ling, Q. and M. Lemmon (2003). "Soft real-time scheduling of networked control systems with dropouts governed by a markov chain". In: *Proceedings of the 2003 American Control Conference, 2003.* Vol. 6, 4845–4850 vol.6. DOI: 10.1109/ACC.2003.1242490.

Linsenmayer, S. and F. Allgöwer (2018). "Performance oriented triggering mechanisms with guaranteed traffic characterization for linear discrete-time systems". In: *European Control Conference (ECC)*, pp. 1474–1479. DOI: 10.23919/ECC.2018.8550568.

Liu, C. and J. Layland (1973). "Scheduling algorithms for multiprogramming in a hard-real-time environment". *Journal of the ACM* **20**:1, pp. 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743.

Liu, C. and J. H. Anderson (2010). "Supporting soft real-time dag-based systems on multiprocessors with no utilization loss". In: *IEEE Real-Time Systems Symposium*, pp. 3–13. DOI: 10.1109/RTSS.2010.38.

Liu, D., X. S. Hu, M. Lemmon, and Q. Ling (2005). "Scheduling tasks with markov-chain based constraints". In: *Euromicro Conference on Real-Time Systems*, pp. 157–166. DOI: 10.1109/ECRTS.2005.27.

Ma, Y., Y. Wang, S. Cairano, T. Koike-Akino, J. Guo, P. Orlik, and C. Lu (2018). "A smart actuation architecture for wireless networked control systems". In: DOI: 10.1109/CDC.2018.8619831.

Mai, H., A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King (2011). "Debugging the data plane with anteater". *SIGCOMM Computer Communication Review* **41**:4. ISSN: 0146-4833. DOI: 10.1145/2043164.2018470.

Mandrioli, C. (2022). *Control-Theoretical Perspective in Feedback-Based Systems Testing*. PhD thesis. Lund University.

Manolache, S., P. Eles, and Z. Peng (2004). "Optimization of soft real-time systems with deadline miss ratio constraints". In: *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 562–570. DOI: 10.1109/RTTAS.2004.1317304.

Mirkin, L. (2004). "On the approximation of distributed-delay control laws". *Systems & Control Letters* **51**:5, pp. 331–342. ISSN: 0167-6911. DOI: 10.1016/j.sysconle.2003.09.010.

Mirkin, L. and Z. J. Palmor (2005). "Control issues in systems with loop delays". *Handbook of networked and embedded control systems*, pp. 627–648.

Nilsson, J. (1998). *Real-Time Control Systems with Delays*. PhD thesis.

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998). "Stochastic analysis and control of real-time systems with random time delays". *Automatica* **34**:1, pp. 57–64. ISSN: 0005-1098. DOI: 10.1016/S0005-1098(97)00170-2.

Nyberg Carlsson, M., N. Vreman, and A. Cervin (2023). "Timing-Robust Control over the Cloud Using On-Line Parametric Optimization". In: *IFAC World Congress*.

Ohlin, M., D. Henriksson, and A. Cervin (2006). *Truetime 1.4 - reference manual*.

Park, P., S. Coleri Ergen, C. Fischione, C. Lu, and K. H. Johansson (2018). "Wireless network design for control systems: a survey". *IEEE Communications Surveys & Tutorials* **20**:2, pp. 978–1013. DOI: 10.1109/COMST.2017.2780114.

Parrilo, P. and A. Jadbabaie (2008). "Approximation of the joint spectral radius using sum of squares". *Linear Algebra and its Applications*.

Pazzaglia, P., L. Pannocchi, A. Biondi, and M. D. Natale (2018). "Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses". In: Altmeyer, S. (Ed.). *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:22. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.10.

Protasov, V. Y. (1996). "The joint spectral radius and invariant sets of the several linear operators". *Fundamentalnaya i Prikladnaya Matematika* **2**. ISSN: 0024-3795.

Rota, G.-C. and W. G. Strang (1960). "A note on the joint spectral radius". In: *Proceedings of the Netherlands Academy*.

Saifullah, A., D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill (2014). "Parallel real-time scheduling of dags". *IEEE Transactions on Parallel and Distributed Systems* **25**:12, pp. 3242–3252. DOI: 10.1109/TPDS.2013.2297919.

Schenato, L. (2009). "To zero or to hold control inputs with lossy links?" *IEEE Transactions on Automatic Control* **54**:5, pp. 1093–1099.

Soudbakhsh, D., L. T. X. Phan, A. M. Annaswamy, and O. Sokolsky (2018). "Co-design of arbitrated network control systems with overrun strategies". *IEEE Transactions on Control of Network Systems* **5**:1, pp. 128–141. DOI: 10.1109/TCNS.2016.2583064.

Stigge, M., P. Ekberg, N. Guan, and W. Yi (2011). "The digraph real-time task model". In: *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 71–80. DOI: 10.1109/RTAS.2011.15.

Tsog, N., S. Mubeen, M. Behnam, M. Sjödin, and F. Bruhn (2021). "Simulation and analysis of in-orbit applications under radiation effects on cots platforms". In: *IEEE Aerospace Conference*, pp. 1–8. DOI: 10.1109/AERO50100.2021.9438255.

Vogt, D., C. Giuffrida, H. Bos, and A. S. Tanenbaum (2015). "Lightweight memory checkpointing". In: *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 474–484. DOI: 10.1109/DSN.2015.45.

Voss, W. (2005). *A comprehensible Guide to Controller Area Network*. 2nd ed. Copperhill Media Corporation, USA. ISBN: 0976511606.

Vreman, N., A. Cervin, and M. Maggio (2021). "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.15.

Vreman, N. and M. Maggio (2019). "Multilayer Distributed Control over 5G Networks: Challenges and Security Threats". In: *Proceedings of the Workshop on Fog Computing and the IoT (Fog-IoT)*. IoT-Fog '19. Association for Computing Machinery, New York, NY, USA. ISBN: 9781450366984. DOI: 10.1145/3313150.3313223.

Vreman, N. and M. Maggio (2023). *"*Stochastic Stability Analysis of Control Systems Subject to Communication and Computation Faults*"*. *Submitted to ACM SIGBED International Conference on Embedded Software (EMSOFT)*.

Vreman, N. and C. Mandrioli (2020). "Evaluation of Burst Failure Robustness of Control Systems in the Fog". In: *2nd Workshop on Fog Computing and the IoT (Fog-IoT)*. Vol. 80. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-95977-144-3. DOI: 10.4230/OASIcs.Fog-IoT.2020.8.

Vreman, N., C. Mandrioli, and A. Cervin (2022a). "Deadline-Miss-Adaptive Controller Implementation for Real-Time Control Systems". In: *IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. DOI: 10.1109/RTAS54340.2022.00010.

Vreman, N., R. Pates, K. Krüger, G. Fohler, and M. Maggio (2019). "Minimizing Side-Channel Attack Vulnerability via Schedule Randomization". In: *IEEE 58th Conference on Decision and Control (CDC)*. DOI: 10.1109/CDC40024.2019.9030144.

Vreman, N., R. Pates, and M. Maggio (2022b). "WeaklyHard.jl: Scalable Analysis of Weakly-Hard Constraints". In: *IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. DOI: 10.1109/RTAS54340.2022.00026.

Vreman, N., P. Pazzaglia, V. Magron, J. Wang, and M. Maggio (2022c). "Stability of Linear Systems Under Extended Weakly-Hard Constraints". *IEEE Control Systems Letters* **6**. DOI: 10.1109/LCSYS.2022.3179960.

Wadleigh, K. and I. Crawford (2000). *Software Optimization for High-performance Computing*. HP Professional. Prentice Hall PTR. ISBN: 9780130170088.

Wang, J., M. Maggio, and V. Magron (2021a). "SparseJSR: A Fast Algorithm to Compute Joint Spectral Radius via Sparse SOS Decompositions". *American Control Conference*.

Wang, J., V. Magron, and J.-B. Lasserre (2021b). "TSSOS: A Moment-SOS hierarchy that exploits term sparsity". *SIAM Journal on Optimization* **31**:1.

WSTS; SIA (2022). *Semiconductor market size worldwide from 1987 to 2022 (in billion u.s. dollars)*. URL: https://www.statista.com/statistics/266973/global-semiconductor-sales-since-1988/ (visited on 2023-01-23).

Xing, L., C. Wen, Z. Liu, H. Su, and J. Cai (2017). "Adaptive compensation for actuator failures with event-triggered input". *Automatica* **85**, pp. 129–136. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2017.07.061.

Xiong, J. and J. Lam (2007). "Stabilization of linear systems over networks with bounded packet loss". *Automatica* **43**:1, pp. 80–87. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2006.07.017.

Yoshimoto, T. and T. Ushio (2011). "Optimal arbitration of control tasks by job skipping in cyber-physical systems". In: *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pp. 55–64. DOI: 10.1109/ICCPS.2011.18.

Zhu, F., C. Zhang, Z. Zheng, and A. Farouk (2021). "Practical network coding technologies and softwarization in wireless networks". *IEEE Internet of Things Journal* **8**:7, pp. 5211–5218. DOI: 10.1109/JIOT.2021.3056580.

Paper I

Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses

Nils Vreman Anton Cervin Martina Maggio

Abstract

Control systems are by design robust to various disturbances, ranging from noise to unmodelled dynamics. Recent work on the weakly hard model—applied to controllers—has shown that control tasks can also be inherently robust to deadline misses. However, existing exact analyses are limited to the stability of the closed-loop system. In this paper we show that stability is important but cannot be the only factor to determine whether the behaviour of a system is acceptable also under deadline misses. We focus on systems that experience bursts of deadline misses and on their recovery to normal operation. We apply the resulting comprehensive analysis (that includes both stability and performance) to a Furuta pendulum, comparing simulated data and data obtained with the real plant. We further evaluate our analysis using a benchmark set composed of 133 systems, which is considered representative of industrial control plants. Our results show the handling of the control signal is an extremely important factor in the performance degradation that the controller experiences—a clear indication that only a stability test does not give enough indication about the robustness to deadline misses.

Originally published in Leibniz International Proceedings in Informatics (LIPIcs) 33rd Euromicro Conference on Real-Time Systems (2021). The mathematical notation has been unified to match the remainder of the thesis. Reprinted with permission.

1. Introduction

Feedback control systems have been used as prime examples of hard real-time systems ever since the term was coined. However, in the past twenty years, it has become increasingly clear that the hard real-time task model is overly strict for most control systems. Requiring that *all* deadlines of a periodic control task must be met can lead to very conservative designs with low utilisation, low sampling rates, and—in the end—worse than necessary control performance. Following this line of reasoning, researchers started looking into task models in which tasks can sporadically miss some deadlines, and defined concepts like the "skip factor" [Koren and Shasha, 1995], i.e., the number of correctly executed jobs that must occur between two failed instances. Task models with failed jobs eventually led to the definition of the weakly hard task model [Bernat et al., 2001], that specify constraints on the sequence of jobs that complete their execution correctly and the ones that miss their deadlines. Adopting the weakly hard model allows a control task to opportunistically execute more frequently, which in general improves reference tracking and disturbance rejection [Kauer et al., 2014; Linsenmayer and Allgower, 2017; Pazzaglia et al., 2018].

A recent industrial survey has shown that practitioners are used to work with systems that experience deadline misses [Åkesson et al., 2020, Questions 14 and 15]. In a significant percentage of cases, these systems are subject to blackout events that can persist for more than ten consecutive task periods. Examples of such events are mode switches in mixed-criticality systems, resets due to hardware faults, security attacks, specific types of cache misses, and connectivity issues in networked control systems. Handling all of these situations by design could require extreme resource over-provisioning.

In this paper we focus precisely on these sporadic system events, which may cause a control task to stall for one or several cycles. To determine the effect of deadline misses on the control system, it is of utmost importance to analyse the physics of the plant and the effect of control signals not being delivered to it. For these systems, stability guarantees have been given on the maximum number of tolerable consecutive deadline misses [Maggio et al., 2020]. These guarantees only consider *stability* of the closed-loop system as the property to be preserved. In this paper, we demonstrate that while stability may be preserved, the control system *performance* may be severely affected by the burst of misses. Performance and stability have been considered simultaneously in the literature. For example, in [Ghosh et al., 2018] a controller is developed that guarantees stability, accepting some level of performance degradation for a given plant. However, we believe that a lot is left open to investigate, especially with respect to general guarantees. In particular, in this paper we aim to understand the effect that the deadline handling strategies jointly have on performance and stability, providing a holistic evaluation. Furthermore, we evaluate our results on both simulated platforms and real control plants. More precisely, we offer the following contributions:

- We propose a new type of weakly hard task model, which specifies a *consecutive* deadline miss interval followed by a minimum *consecutive* deadline hit (recovery) interval. This model is crucial to properly assess the performance effect of a burst of deadline misses, as the ones reported by practitioners [Åkesson et al., 2020].
- We provide an analysis methodology for stability and performance of control tasks executing under this task model using a variety of implementation choices to handle deadline misses (`Kill` vs. `Skip`, `Zero` vs. `Hold`). In particular, we separately consider the two cases in which a miss pattern is repeated (which fits an increased workload situation—for example due to a different mode of execution), and in which it is not possible to specify constraints on the repetition of the miss pattern.
- We compare experimental results obtained with a real process—a Furuta pendulum that is stabilised in the upright position—with simulation results based on a linear model of the same process, using the same controller. This shows that simulated data is representative enough to draw conclusions on the controller performance, despite unmodelled nonlinear dynamics and noise.
- We present the result of a large scale evaluation campaign of commonly used controllers on a benchmark of 133 industrial plants. From this evaluation we conclude that the choice of actuation strategy (i.e., what to do with the control signal when a miss occurs) affects control performance significantly more than the choice of deadline handling strategy (i.e., what to do with the control task when a miss occurs).

The rest of this paper is outlined as follows. In Section 2 we give a brief overview of related work. In Section 3 we present relevant control theory and introduce the stability and performance concepts. Section 4 describes the weakly hard task models and the strategies that are commonly used to handle deadline misses. Section 5 presents our extension to the weakly hard task model, and the corresponding stability and performance analysis. Section 6 presents our experimental results, and Section 7 concludes the paper.

2. Related Work

The work presented in this paper is closely related to two broad research areas, namely, the analysis of (i) weakly hard systems and (ii) fault-tolerant control systems.

Weakly Hard Systems: Deadline misses can be seen as sporadic events caused by unforeseen delays in the system. Such delays could for instance be induced by overload activations [Xu et al., 2015; Hammadeh et al., 2014] or cache misses [Altmeyer and Davis, 2014; Davis et al., 2013]. The idea behind weakly hard analysis

is that deadline misses are permitted under predefined constraints. Such systems have been analysed extensively from a real-time scheduling perspective [Bernat and Burns, 1997; Caccamo and Buttazzo, 1997; Choi et al., 2019; Hammadeh et al., 2019]. The weakly hard models have gained traction in the research community as a tool to understand and analyse systems with sporadic faults [Soudbakhsh et al., 2013; Bund and Slomka, 2014; Frehse et al., 2014; Bund and Slomka, 2015; Hammadeh et al., 2017b; Hammadeh et al., 2017a; Sun and Natale, 2017; Ahrendts et al., 2018; Soudbakhsh et al., 2018; Pazzaglia et al., 2018; Gaukler et al., 2019]. In a recent paper, Gujarati et al. [Gujarati et al., 2019] analysed and compared different methods for estimating the overall reliability of control systems using the weakly hard task model. Furthermore, the authors of [Natarajan et al., 2019] proposed a toolchain for analysing the strongest, satisfied weakly hard constraints as a function of the worst-case execution time.

Fault-Tolerant Control Systems: Real-time systems are sensitive to faults. Due to their safety-critical nature, it is arguably more important to guarantee fault-tolerance with respect to other classes of systems. Some of these faults can be described using the weakly hard model. Due to the nature of control systems, special analysis techniques can combine fault models and the physical characteristics of systems.

Fault-tolerance has been investigated in many of its aspects, e.g., fault-aware scheduling algorithms [de Niz et al., 2013; Caccamo et al., 2000] and the analysis of systems with unreliable components [Khosravi et al., 2015]. Furthermore, restart-based design [Abdi et al., 2017a; Abdi et al., 2019] has been used as a technique to guarantee resilience. The fault models are frequently assumed to target overload-prone systems, or systems with components subject to sporadic failures. Bursts of faults have been observed to affect real systems [Chen et al., 2015; Vreman and Mandrioli, 2020]. Gujarati et al. [Gujarati et al., 2018] proposed an analysis method for networked control systems that uses active replication and quantifies the resilience of the control system to stochastic errors. Maggio et al. [Maggio et al., 2020] developed a tool for determining the stability of a control system where the control task behaves according to the weakly hard model. From the control perspective, there has been extensive research into both analysis and mitigation of real-time faults in feedback systems [Ramanathan, 1997; Goswami et al., 2014; Ghosh et al., 2018]. Very often, this research produced tools to analyse the effect of computational delays [Cervin et al., 2019] and of choosing specific scheduling policies or parameters [Palopoli et al., 2000; Cervin, 2005], possibly including deadline misses. In a few instances, researchers looked at how to improve the performance of control systems in conjunction with scheduling information [Buttazzo et al., 2007]. One such effort analyses modifications to the code of classic and simple control systems to handle overruns that reset the period of execution of the control task [Pazzaglia et al., 2021]. Abdi et al. [Abdi et al., 2017b] proposed a control design method for safe system-level restart, mitigating unknown faults during runtime execution, while keeping the system inside a safe operating space. Pazzaglia et al. [Pazzaglia

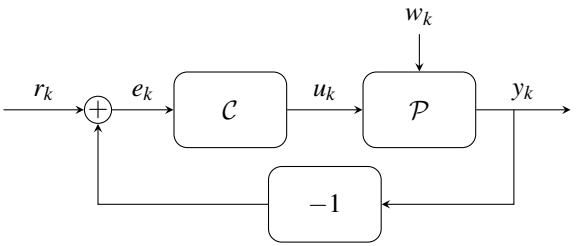


Figure 1. Control loop: The reference value r_k is compared with the output y_k of the plant $\mathcal{P}$. The control error $e_k = r_k - y_k$ is used by the controller $\mathcal{C}$ to compute the value of the control signal u_k. The plant is disturbed by the stochastic process w_k.

et al., 2019] used the scenario theory to derive a control design method accounting for potential deadline misses, and discussed the effect of different deadline handling strategies. Linsenmayer et al. [Linsenmayer et al., 2020] worked on the stabilisation of weakly-hard linear control systems for networked control systems, with some extension for nonlinear systems [Hertneck et al., 2019]. In the considered setup, faults compromise network transmissions, but do not interfere with the controller computation (assuming that the computation is triggered). The work also focused on stability, with no control performance evaluation.

To the best of our knowledge, no previous work has devised a combined stability and performance analysis to understand how faults (even when they can be tolerated) affect the plant that should be controlled when different deadline handling strategies are used.

3. System Behaviour in Nominal Conditions

In this section, we introduce the relevant control background needed for the remainder of the paper, and we detail how the controller and the system behave under normal operation.

3.1 Plant Model

We first describe the model we use for the object we are trying to control. In control terms—mostly due to historical reasons—this object is called a *plant*. Examples range from a pendulum that we would like to stabilise in the upward position, to a chemical dilution process, to the distribution of workload in a datacenter.

Plants are usually modelled as continuous- or discrete-time dynamical systems. All real-world plants are nonlinear, but for control design purposes they are often linearised around their operating points. Around such a point, the resulting model becomes a Linear Time-Invariant (LTI) system. In this paper, we restrict our analysis to discrete-time LTI systems, because we investigate controllers implemented

with fixed-rate sampling and actuation in digital electronics. To design and analyse these systems, we use the discrete-time counterpart of the continuous-time physical model, which can be obtained with standard techniques [Åström and Wittenmark, 1997].

We consider a plant $\mathcal{P}$ described in state-space form:

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

In (1), k counts the discrete instants that represent the plant's sampling points. We assume periodic sampling; the time between two consecutive samples k and $k+1$ is fixed and equal to sampling period T. In the equation, x_k is a column vector with n_x elements. These elements represent the state variables that account for, e.g., the storage of mass, momentum, and energy. Similarly, u_k is a column vector with n_u elements. These values represent the inputs that affect the dynamics of the plant. We also consider w_k, a column vector with n_u elements. The term w_k represents an unknown load disturbance, modelled as a stationary stochastic process with known properties. Finally, y_k is a column vector with n_y elements, that represents the measurements that are taken from our plant. The matrices A (size $n_x \times n_x$), B (size $n_x \times n_u$), C (size $n_y \times n_x$), D (size $n_y \times n_u$), and W (size $n_x \times n_u$) characterise the dynamics of the plant.

3.2 Controller Model

The plant $\mathcal{P}$ is controlled by a periodically executing controller $\mathcal{C}$ with implicit deadlines, i.e., the deadline of each task instance (job) coincides with the next task activation. We consider the class of all linear controllers with a one-step delay between sampling and actuation.[1] In other words, we consider all the controllers that can be written as linear systems, according to the following state-space equation:

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

Here, z_k is a column vector with n_z elements that represents the state of the controller. The input of the controller is e_k, a vector of n_y elements. Each element in the vector is the error between the corresponding plant output and its reference value ($e_k = r_k - y_k$, where r_k represents the reference values for the plant outputs).

[1] One-step delay controllers are controllers in which a control signal is computed in the k-th interval and actuated at the beginning of the $k+1$-th period. In the real-time systems jargon, one-step delay controllers are often referred to as controllers that follow the Logical Execution Time (LET) paradigm [Kirsch and Sokolova, 2012; Ernst et al., 2018]. From the real-time perspective, implementing the controller following the LET paradigm improves the timing predictability. From the control perspective, one-step delay controllers reduce activation jitter and allows the engineer to neglect time-varying computational delays.

Finally, u_k is a vector of n_u elements, that encodes the output of the controller, which is connected to the plant input vector. The matrices F (size $n_z \times n_z$), G (size $n_z \times n_y$), H (size $n_u \times n_z$), and K (size $n_u \times n_y$) characterise the dynamics of the controller. For every task activation, the controller first applies the value of u_k that was computed by the previous job and then reads the inputs r_k and y_k. It then calculates the values of z_{k+1} and u_{k+1} that will be used in the next iteration.

The analysis methodology presented in the remainder of this paper is valid for *all* linear controllers. The class of linear controllers includes some of the most frequently used controllers in industry, in particular proportional and integral (PI), proportional, integral, and derivative (PID), lead–lag compensators, and linear-quadratic-Gaussian (LQG) controllers. Although the performance analysis is presented for the time-invariant case, the formulas are valid also for systems with time-varying matrices. Hence, it is possible to analyse plants and controllers that transition between different local linear models.

3.3 Closed-Loop System Dynamics

We now analyse the closed-loop system shown in Figure 1. Combining the dynamical models from (1) and (2), we obtain matrices that represent the closed-loop system. We denote the state vector of the closed-loop system with $\tilde{x}_k = \left[x_k^{\mathrm{T}}, z_k^{\mathrm{T}}, u_k^{\mathrm{T}}\right]^{\mathrm{T}}$, where T is the transpose operator. In this way, we obtain a system that has the vectors r_k and w_k as input, and is described by

$$\mathcal{S}_{\mathrm{cl}} : \begin{cases} \tilde{x}_{k+1} = \Phi \tilde{x}_k + \Gamma_r r_k + \Gamma_w w_k \\ y_k = \tilde{C} \tilde{x}_k, \end{cases} \tag{3}$$

where the closed-loop state matrix Φ is

$$\Phi = \begin{bmatrix} A & 0_{n_x \times n_z} & B \\ -GC & F & -GD \\ -KC & H & -KD \end{bmatrix}, \tag{4}$$

the input matrices Γ_r and Γ_w are

$$\Gamma_r = \begin{bmatrix} 0_{n_x \times n_y} \\ G \\ K \end{bmatrix}, \quad \Gamma_w = \begin{bmatrix} W \\ 0_{n_z \times n_u} \\ 0_{n_u \times n_u} \end{bmatrix}, \tag{5}$$

and the output matrix $\tilde{C}$ is

$$\tilde{C} = \begin{bmatrix} C & 0_{n_x \times n_z} & D \end{bmatrix}. \tag{6}$$

Figure 2 shows the graphical representation of the closed-loop system $\mathcal{S}_{\mathrm{cl}}$, with input and output signals.

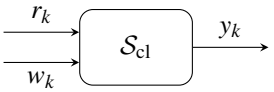


Figure 2. Closed-loop system rewritten as a new linear system $\mathcal{S}_{\text{cl}}$. The resulting system has two inputs, r_k and w_k and one output. The feedback loop shown in Figure 1 is hidden inside $\mathcal{S}_{\text{cl}}$.

Stability To assess the stability of the closed-loop system under normal operation, it is sufficient to check the eigenvalues of the state matrix. According to the Schur stability criterion [Åström and Wittenmark, 1997], if the eigenvalues of Φ lie within the unit disc, then the system is asymptotically stable. Formally, a closed-loop system is Schur stable if and only if

$$\max_i |\text{eig}_i(\Phi)| < 1, \tag{7}$$

where $\text{eig}_i(\Phi)$ is a function that returns the i-th eigenvalue of Φ.

If the system dynamics change at runtime (e.g., in the case of a lost sample, unexpected delay, or computational problem), Schur stability is no longer a sufficient stability criterion. Instead, *switching stability analysis* can be employed to check the stability of a system with alternating dynamics [Jungers, 2009]. There has been a lot of research on the switching stability analysis, with multiple tools developed in order to simplify the analysis. Two main methods are employed: (i) the search for a common Lyapunov function, e.g., as done in [Linsenmayer and Allgower, 2017], (ii) the computation of the Joint Spectral Radius (JSR), e.g., as done in [Maggio et al., 2020; Vankeerberghen et al., 2014].

Performance Alongside stability, it is important to look at the *performance* of the closed-loop system. Performance can be defined in different ways, often depending on the application [Åström and Hägglund, 2006]. Whichever way is chosen, a common way to quantify performance is to define a cost function and evaluate the cost function during the execution of the controller. In our work, we use a quadratic cost function

$$J_k = \mathbb{E}\left[e_k^{\mathrm{T}} Q_e e_k + u_k^{\mathrm{T}} Q_u u_k\right]. \tag{8}$$

The cost function penalises deviations from the reference value as well as usage of the control signal. $\mathbb{E}[\cdot]$ denotes expected value, and the positive semidefinite weighting matrices Q_e (size $n_y \times n_y$) and Q_u (size $n_u \times n_u$) weigh the different terms against each other. A small cost value means that the controller successfully makes the error approach zero, using a small control signal.

If the stochastic properties of the external signals r_k and w_k are known, it is possible to calculate the value of the cost function analytically. For simplicity and without loss of generality, we will henceforth assume that $r_k = 0$ (i.e., we want to

regulate the output to zero) and that w_k is a zero-mean Gaussian white noise process with variance $R = \mathbb{E}\left[w_k w_k^{\mathrm{T}}\right]$. More elaborate disturbance models can be realised by adding extra states in the plant model.

We now detail how to evaluate (8). Let P_k denote the covariance of the closed-loop state vector at time k,

$$P_k = \mathbb{E}\left[\tilde{x}_k \tilde{x}_k^{\mathrm{T}}\right]. \tag{9}$$

The state covariance evolves according to

$$P_{k+1} = \Phi P_k \Phi^{\mathrm{T}} + \Gamma_w R \Gamma_w^{\mathrm{T}}. \tag{10}$$

Given P_k, we can calculate the cost for time step t as

$$J_k = \mathbb{E}\left[\tilde{x}_k^{\mathrm{T}} Q \tilde{x}_k\right] = \mathrm{tr}\left(P_k Q\right), \tag{11}$$

where tr computes the trace of the matrix, and

$$Q = \begin{bmatrix} C^{\mathrm{T}} Q_e C & 0_{n_x \times n_z} & 0_{n_x \times n_u} \\ 0_{n_z \times n_x} & 0_{n_z \times n_z} & 0_{n_z \times n_u} \\ 0_{n_u \times n_x} & 0_{n_u \times n_z} & Q_u \end{bmatrix} \tag{12}$$

is the total cost matrix. The stationary cost of the system is defined as J_∞. This is the cost that the system converges to when operating under normal conditions:

$$J_\infty = \lim_{k \to \infty} J_k. \tag{13}$$

This means that there exists an instant $\bar{k}$ for which J_k reaches a value arbitrarily close to the steady-state value J_∞, or $\forall \varepsilon,\ \exists \bar{k}$ s.t. $\forall k > \bar{k},\ |(J_k - J_\infty)/J_\infty| < \varepsilon$.

4. System Behaviour with Deadline Misses

The analysis above holds when the control task meets all its deadlines. However, the presence of deadline misses changes the behaviour of the system. The stability of controllers with a number of consecutive deadline misses has been investigated in [Maggio et al., 2020]. The results of this investigation attested that, due to their inherent robustness, many control systems can withstand at least a small number of consecutive misses.

To analyse the system, we need to clarify three aspects about the miss behaviour:

(i) What happens to the control signal.

(ii) What happens to the control task.

(iii) The computational model used for the analysis (how many deadlines can we miss, and in what pattern).

For the first item, the actuator can either output a `Zero` ($u_k = 0_{n_u \times 1}$), or `Hold` the previous value ($u_k = u_{k-1}$). The choice depends on both the plant dynamics and on the controller, as no strategy in general dominates the other one [Schenato, 2009]. For controllers with integral action, it makes sense to hold the previous control value, under the presumption that the system is still disturbed and that a non-zero control signal is needed to keep the plant close to its operating point. On the other hand, the `Zero` strategy may be preferred for plants with unstable or integrator dynamics, where outputting a zero control action may be the safer option.

Considering the second item, at least three different strategies can be employed to deal with a control task that misses its deadline [Cervin, 2005]: (i) `Kill`, (ii) `Skip`, (iii) and `Queue`(σ) ($\sigma \in \{1,2,3,\ldots\}$). When the `Kill` strategy is used, the job that missed its deadline is terminated, its changes are rolled back, and the next job is released. Following the `Skip` strategy, the job that missed its deadline continues its execution. No new control task jobs are released until the currently running one completes its execution. `Queue`(σ) behaves similarly to `Skip` in allowing the current job to complete execution, but also allows the activation of new jobs (the queue of active jobs holds up to the most recent σ instances of the control task). In this paper we only analyse `Kill` and `Skip`. In fact, the results presented in [Cervin, 2005; Maggio et al., 2020] suggest that `Queue`(σ) is not a feasible strategy to handle misses. The presence of two or more active jobs in the same period creates a chain effect that is hard to recover from and that deteriorates stability and performance.

The last item refers to models of computation. The weakly hard task model [Hamdaoui and Ramanathan, 1995; Bernat et al., 2001] is usually considered expressive enough to analyse the behaviour of tasks that miss their deadlines. The authors of [Bernat et al., 2001] propose four definitions for a weakly hard real-time task τ:

DEFINITION 1—WEAKLY HARD TASK MODELS [BERNAT ET AL., 2001]
A task τ may satisfy any of these four weakly hard constraints:

(i) $\tau \vdash \binom{x}{\ell}$*: there are at least x hits for every ℓ jobs,*

(ii) $\tau \vdash \overline{\binom{x}{\ell}}$*: there are at most x misses for every ℓ jobs,*

(iii) $\tau \vdash \left\langle {x \atop \ell} \right\rangle$*: there are at least x consecutive hits for every ℓ jobs,*

(iv) $\tau \vdash \overline{\left\langle {x \atop \ell} \right\rangle}$*: there are at most x consecutive misses for every ℓ jobs.*

There has been a lot of research on the second model, often also called *m-K* model [Koren and Shasha, 1995; Ramanathan, 1997; Soudbakhsh et al., 2013; Bund and Slomka, 2014; Frehse et al., 2014; Bund and Slomka, 2015; Hammadeh et al., 2017b; Hammadeh et al., 2017a; Sun and Natale, 2017; Ahrendts et al., 2018; Soudbakhsh et al., 2018; Pazzaglia et al., 2018; Pazzaglia et al., 2019; Gaukler et al.,

2019] (with m being the maximum number of misses in a window of K activations). Recently there has also been an analysis of the stability of control systems when the control task behaves according to the fourth model [Maggio et al., 2020].

If the misses are due to faults or security attacks, usually the control task experiences an interval of consecutive misses. When the fault is resolved, the control task starts hitting its deadlines again. From the performance standpoint, a consecutive number of misses degrades the control quality. We are interested in what degradation is acceptable and how much time should occur between two potential failures. Specifically, we look at how many deadline hits should follow a given number of consecutive misses for the system to *recover*. None of the four models above allow us to formulate this requirement (as they specify either consecutive hits or misses but not both), which leads us to introduce a different weakly hard model of computation, together with its analysis, in Section 5.

5. Burst Interval Analysis

In this section, we analyse the stability and performance of a real-time control system that experiences bursts of deadline misses. Section 5.1 introduces the fault model, Section 5.2 derives the control system behaviour subject to different real-time policies and delves into both the stability and performance analysis.

5.1 Fault Model

Faults can happen during the normal execution of tasks on a platform. Informally, as a result of a fault, tasks miss their deadlines. When the fault is resolved, then the original situation is recovered (possibly after a transient initial phase).

Specifically, given a system $\mathcal{S}_{\text{cl}}$, we define a *burst interval* $\mathcal{M}$ as an interval of controller activations in which the control task executing $\mathcal{C}$ consecutively misses m deadlines, regardless of the strategy used to handle the misses. We assume that the burst interval $\mathcal{M}$ is followed by a *recovery interval* $\mathcal{R}$, defined as an interval in which the control task consecutively hits n deadlines.

During the burst interval, the deadline misses of the control task are handled using a *deadline handling strategy* $\mathcal{D}$ (`Kill`, `K`, or `Skip`, `S`). The control signal u_k is selected in accordance with the *actuation strategy* $\mathcal{A}$ (`Zero`, `Z`, or `Hold`, `H`). We denote the combination of $\mathcal{D}$ and $\mathcal{A}$ with $\mathcal{H} = (\mathcal{D}, \mathcal{A})$. For example $\mathcal{H}$ could be `SZ` to indicate that the `Skip` deadline handling strategy is paired with the `Zero` actuation strategy. The system *recovers* once it operates close to steady-state.

From an industrial viewpoint, the proposed fault model is highly relevant. The common approach is to treat faults as pseudo-independent events adhering to predefined constraints on their incidence rate [OConnor and Kleyner, 2012; Montgomery, 2009; Khosravi et al., 2017]. However, during the operation of a control system, faults can be caused by events like network connection problems (e.g., cutting the connection between the sensor and the controller), security attacks, con-

tention on resources. Studies in the automotive sector, for example, indicate that deadline misses can occur in bursts [Quinton et al., 2014; Xu et al., 2015]. In these cases, the controller does not execute properly for a given amount of time (e.g., until the connection is restored, the attack is terminated, or the resource contention is reduced). The analysis methods we propose allow us to address such situations and to provide tighter bounds on the closed-loop stability and performance than under the previously proposed weakly hard models. Moreover, following a burst interval, we are interested in analysing the length of the recovery interval $\mathcal{R}$ that is needed to return to normal operation under each implementation strategy $\mathcal{H}$. Hence, we here extend the weakly hard models of computation with a fifth alternative and then devote the remainder of the paper to its analysis.

DEFINITION 2—WEAKLY HARD FAULT MODEL WITH BURST OF MISSES
A real-time task τ may satisfy the weakly hard task model

(v) $\tau \vdash \left\{{m \atop \ell}\right\}$*: there are at most m consecutive misses, followed by* $\ell - m$ *consecutive hits for every* ℓ *jobs.*

This means that a real-time task τ behaves according to the model $\tau \vdash \left\{{m \atop \ell}\right\}$, if, whenever τ experiences a burst interval $\mathcal{M}$ consisting of m consecutive deadline misses, it is always followed by a recovery interval $\mathcal{R}$ consisting of $n = \ell - m$ consecutive deadline hits.

5.2 Closed-Loop System Dynamics

In this section we derive the system dynamics for a closed-loop control system under the assumption that we enter a burst interval of length m after time instant k, and after m deadline misses we start completing the control job in time.

Normal Operation: Under *normal operating conditions* the system is not experiencing any deadline misses. In other words, the system evolves according to the closed-loop system dynamics (3).

`Kill&Zero:` If a control task deadline miss occurs at time instant k, the plant states x_k still evolve as normal. However, the controller terminates its execution prematurely by killing the job, thus not updating its states ($z_{k+1} = z_k$). The controller output is determined by the actuation strategy and is here zero ($u_{k+1} = 0$). Now, consider a burst interval of length m after time instant k. Recalling that $\tilde{x}_k = [x_k{}^{\mathrm{T}}\, z_k{}^{\mathrm{T}}\, u_k{}^{\mathrm{T}}]^{\mathrm{T}}$, we can write the evolution of the closed-loop system for the sequence of m deadline misses followed by a single deadline hit as the product of a matrix representing the behaviour of the system for a hit and a matrix representing the behaviour in case of miss elevated to the power of m to indicate m steps of the system evolution.

The resulting closed-loop system in state-space form is

$$\begin{bmatrix} x_{k+m+1} \\ z_{k+m+1} \\ u_{k+m+1} \end{bmatrix} = \underbrace{\Phi \begin{bmatrix} A & 0_{n_x \times n_z} & B \\ 0_{n_z \times n_x} & I & 0_{n_z \times n_u} \\ 0_{n_u \times n_x} & 0_{n_u \times n_z} & 0_{n_u \times n_u} \end{bmatrix}^m}_{\Phi_{\mathtt{KZ}}(m)} \begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix}, \tag{14}$$

where $\Phi_{\mathtt{KZ}}(m)$ represents the system matrix for m misses under the `Kill&Zero` strategy, followed by a single hit (the matrix Φ that is multiplied to the left of the equation). The matrix Φ is the same specified in (4), and represents the first hit that follows the m misses, hence, we determine how $\tilde{x}_k$ influences $\tilde{x}_{k+m+1}$ (m misses and one hit).

`Kill&Hold`: Changing the actuation strategy to `Hold`, slightly alters the system matrix we derived for the `Kill&Zero` case. The plant states x_k evolve as normal and the control states z_k are still not updated ($z_{k+1} = z_k$). However, due to the change in actuation strategy, the last actuated value is instead held ($u_{k+1} = u_k$). The resulting closed-loop state-space form can be seen in (15), where $\Phi_{\mathtt{KH}}(m)$ is used to represent the system matrix for m misses under the `Kill&Hold` strategy and matrix Φ is specified in (4).

$$\begin{bmatrix} x_{k+m+1} \\ z_{k+m+1} \\ u_{k+m+1} \end{bmatrix} = \underbrace{\Phi \begin{bmatrix} A & 0_{n_x \times n_z} & B \\ 0_{n_z \times n_x} & I & 0_{n_z \times n_u} \\ 0_{n_u \times n_x} & 0_{n_u \times n_z} & I \end{bmatrix}^m}_{\Phi_{\mathtt{KH}}(m)} \begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix} \tag{15}$$

`Skip&Zero`: When the control task misses a deadline under the `Skip` strategy, the job missing the deadline is allowed to continue its execution until completion. However, no subsequent job of the control task is released until the current job has finished executing. If the currently active job terminates during period k, the next control job is released at the start of the $k+1$-th period. We can then write the evolution of the system where the control job experiences m misses before completing its execution, meaning that there is a subsequent hit that uses old information for the error measurements. While the controller executed only once to completion, the plant evolved for $m+1$ steps. The resulting closed-loop state-space form can be seen in (16), where $\Phi_{\mathtt{SZ}}(m)$ is used to represent the system matrix under the `Skip&Zero` strategy for m misses and one completion using old measurements.

$$\begin{bmatrix} x_{k+m+1} \\ z_{k+m+1} \\ u_{k+m+1} \end{bmatrix} = \underbrace{\begin{bmatrix} A^{m+1} & 0_{n_x \times n_z} & A^m B \\ -GC & F & -GD \\ -KC & H & -KD \end{bmatrix}}_{\Phi_{\mathtt{SZ}}(m)} \begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix} \tag{16}$$

`Skip&Hold`: Similar to `Skip&Zero`, one job finishes execution after m consecutive misses. However, the actuation strategy holds the previous control value during

the entire burst interval. Therefore, the plant evolution is affected by a cumulative sum over the prior control values. The resulting closed-loop state-space form can be seen in (17), where $\Phi_{\texttt{SH}}(m)$ is used to represent the system matrix for m misses under the `Skip&Hold` strategy.

$$\begin{bmatrix} x_{k+m+1} \\ z_{k+m+1} \\ u_{k+m+1} \end{bmatrix} = \underbrace{\begin{bmatrix} A^{m+1} & 0_{n_x \times n_z} & \sum_{i=0}^{m} A^i B \\ -GC & F & -GD \\ -KC & H & -KD \end{bmatrix}}_{\Phi_{\texttt{SH}}(m)} \begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix} \tag{17}$$

Equations (14)–(17) are inspired by the analysis in [Maggio et al., 2020], but we have we introduced two generalisations. The first one is that our controller is specified as a general state-space system; therefore our method is able to address *all* linear controllers. The second generalisation is that we could include estimates of the plant states in the controller. We can thus properly handle the presence of an observer.[2] Furthermore, we simplify the calculations by reducing the number of states $\tilde{x}_k$ of the closed-loop matrices.

Stability We now describe how the system matrices above can be used to analyse stability. Recall that a closed-loop control system is stable if and only if the (fixed) system matrix Φ is Schur stable. This criterion is also valid for cyclic patterns, where Φ represents the product of all closed-loop state matrices experienced in a full burst–recovery cycle. Hence, we can search for the shortest recovery interval length n such that

$$\max_i \left| \text{eig}_i \left(\Phi^{n-1} \Phi_{\mathcal{H}}(m) \right) \right| < 1, \quad \mathcal{H} \in \{\texttt{KZ}, \texttt{KH}, \texttt{SZ}, \texttt{SH}\}. \tag{18}$$

Recall that $\Phi_{\mathcal{H}}(m)$ already includes one hit, thus the left multiplication with Φ^{n-1}. This is a sufficient condition and not necessary, meaning that a miss occurring during the recovery interval does not immediately imply that the closed-loop system is destabilised. We summarise the analysis in the following definition.

DEFINITION 3—STATIC-CYCLIC STABILITY ANALYSIS
We denote the stability analysis from (18) *with the term* static-cyclic stability analysis*. The system under analysis cycles through a sequence of m misses followed by a sequence of n hits, indefinitely.*

The static-cyclic analysis assumes a repeating burst–recovery cycle with no interruptions. This works well for instance in case the misses are due to a permanent

[2] In [Maggio et al., 2020] the controller state is specified as part of the plant (e.g., when the proportional and integral controller is introduced). This implies that the state is computed although the controller did not execute. Our formulation fixes this by separating the plant execution and the controller states.

overload condition caused by a mode switch (for example from low to high criticality mode in mixed-critical systems). However, the setting is not very general. To foster generality, we complement the stability evaluation with a less restrictive stability analysis, based on the proposed task model in Definition 2.

DEFINITION 4—MISS-CONSTRAINED STABILITY ANALYSIS
To guarantee miss-constrained stability*, a system has to be stable under arbitrary switching between all the possible m realisations (i.e., closed-loop matrices) that comply with all task models* $\tau \vdash \left\{{}^{m_\subset}_{\ell}\right\}, m_\subset \in \{1,\ldots,m\}$ *and also include the case in which the system does not miss deadlines.*

In other words, a system is miss-constrained stable if and only if it is stable under arbitrary switching of the closed-loop matrices in the set

$$\left\{\Phi^{\ell-1}\Phi_{\mathcal{H}}(1), \Phi^{\ell-2}\Phi_{\mathcal{H}}(2), \ldots, \Phi^{\ell-m}\Phi_{\mathcal{H}}(m), \Phi\right\}. \tag{19}$$

Switching stability is unfortunately quite involved.[3] However, many excellent tools have been developed to simplify this analysis (e.g., `MJSR` [Maggio et al., 2020] or the `JSR toolbox` [Vankeerberghen et al., 2014] for MATLAB).

Performance We now show how the cost function in Equation (11) can be used as a time-varying performance metric. Before a burst interval, we assume that the system is in the neighbourhood of its steady-state covariance P_∞ and performance J_∞.

When a burst interval of m missed deadlines occurs, the system will be disrupted and its covariance matrix will evolve according to

$$P_{k+m+1} = \Phi_{\mathcal{H}}(m) P_k \left(\Phi_{\mathcal{H}}(m)\right)^{\mathrm{T}} + \Phi^{j_n} R_w \left(\Phi^{j_n}\right)^{\mathrm{T}}, \tag{20}$$

where

$$\begin{aligned} R_w &= \begin{bmatrix} \sum_{i=0}^{j_m} A^i W R W^{\mathrm{T}} (A^i)^{\mathrm{T}} & 0_{n_x \times n_z + n_u} \\ 0_{n_z+n_u \times n_x} & 0_{n_z+n_u \times n_z+n_u} \end{bmatrix}, \\ j_m &= \begin{cases} m-1 & \text{if } \mathcal{D} = \mathtt{K}\,(\mathtt{Kill}), \\ m & \text{if } \mathcal{D} = \mathtt{S}\ (\mathtt{Skip}), \end{cases} \\ j_n &= \begin{cases} 1 & \text{if } \mathcal{D} = \mathtt{K}\,(\mathtt{Kill}), \\ 0 & \text{if } \mathcal{D} = \mathtt{S}\ (\mathtt{Skip}). \end{cases} \end{aligned} \tag{21}$$

A and W are matrices from the plant evolution in (1), R is the noise intensity from (10), and Φ is the closed-loop matrix from (4). The cost will simultaneously

[3] We have devoted some research effort into the investigation of a suitable stability analysis for control tasks subject to a set of weakly-hard constraints (of the type presented in Defintion 1). A summary of our findings can be found at `https://arxiv.org/abs/2101.11312`.

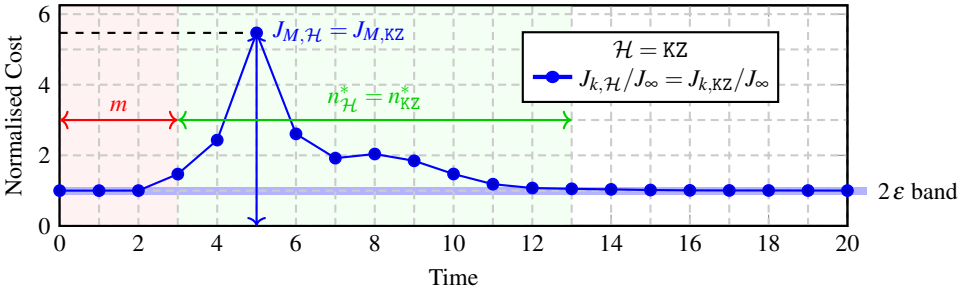


Figure 3. Illustration of normalised cost (J_k/J_∞), performance recovery interval $n^*_{\mathcal{H}}$ and maximum normalised cost $J_{M,\mathcal{H}}$ on a data trace. The example uses $\mathcal{H} =$ `Kill&Zero` and $\varepsilon = 0.1$.

change following (11). In the recovery interval, the covariance is again governed by the normal closed-loop evolution described in (10). The system is said to have recovered once the cost is arbitrarily close to the steady-state cost. We evaluate this as

$$\left| \frac{J_\infty - J_k}{J_\infty} \right| < \varepsilon, \tag{22}$$

where $\varepsilon > 0$ is the *recovery threshold.*

DEFINITION 5—PERFORMANCE RECOVERY INTERVAL
We define the recovery length interval $n^*_{\mathcal{H}}$ as the smallest n such that (22) *is satisfied for all $k \geq n$ when using $\mathcal{H}$ to handle deadline misses.*

DEFINITION 6—MAXIMUM NORMALISED COST
We denote the maximum normalised cost of the system by

$$J_{M,\mathcal{H}} = \max_k \frac{J_{k,\mathcal{H}}}{J_\infty}, \tag{23}$$

where $J_{k,\mathcal{H}}$ is the cost computed according to (11) *when using $\mathcal{H}$ to handle the deadline misses.*

Figure 3 gives a graphical representation of $n^*_{\mathcal{H}}$ and $J_{M,\mathcal{H}}$ for an execution trace in which the controller experiences 3 misses and uses `Kill&Zero` as strategy $\mathcal{H}$.

Compared to the stability analysis, the performance analysis also takes into account state deviations and uncertainty due to disturbances. In Section 5.2 we used the system dynamics to analyse the stability of the system. The disturbance term w_k was neglected as it does not influence the system stability. However, its presence (as the presence of any disturbance) changes the dynamic behaviour of the system. For the performance metric, the state covariance matrix P_k evolves according to both

the noise intensity and the system dynamics (20). The result is that the performance analysis provides us with a conservative (but more realistic) recovery interval, that takes system uncertainties into consideration.

To find the length of the recovery interval, we evolve the state covariance during a burst interval, using a specific strategy $\mathcal{H}$ according to (20). Thereafter, the state covariance is evolved under normal operation, according to (10), until (22) is satisfied, allowing us to find the performance recovery interval $n^*_{\mathcal{H}}$.

6. Experimental Results

In this section, we apply the analysis presented in Section 5 to a set of case studies, analysing stability and performance. We first present detailed results with a Furuta pendulum, both in simulation and with real hardware, using the same controller. The simulated results are compared to the real physical plant. This shows that the performance analysis does capture the important trends for real control systems. We then present some aggregate results obtained with a set of 133 different plants from a control benchmark. One noteworthy aspect is that the Furuta pendulum model is linearised for the control design and the pendulum stabilised around an unstable equilibrium—the top position—while the control benchmark includes (by design) stable systems. The difference between simulation results and real experiments for stable linear systems should in principle be smaller than for unstable nonlinear systems, making our pendulum the ideal stress test for the similarity of simulated and real data.

6.1 Furuta Pendulum

We here analyse the behaviour of a Furuta pendulum [Furuta et al., 1992], a rotational inverted pendulum in which a rotating arm is connected to a pendulum. The rotation of the arm induces a swing movement on the pendulum. The pendulum has two equilibria: a stable position in which the pendulum is downright, and an unstable position in which the pendulum is upright. Our objective is to keep the pendulum in the up position, by moving the rotating arm.

The Furuta pendulum is a highly nonlinear process. In order to design a control strategy to keep the pendulum in the top position, it is necessary to linearise the dynamics of the system around the desired equilibrium point. We consider this as a stress test to check the divergence between simulation results and real hardware results, because of the instability of the equilibrium and the nonlinearity of the dynamics. In fact, the controller necessarily acts with information that is valid only around the upright position, and there is only a range of states in which the linearised model closely describes the behaviour of the physical plant.

We design a linear-quadratic regulator (LQR) to control the plant. Every $T = 10\,\text{ms}$ the plant is sampled and the control signal is actuated. Based on state-of-the-art models [Cazzolato and Prime, 2011] and on our control design, the plant model

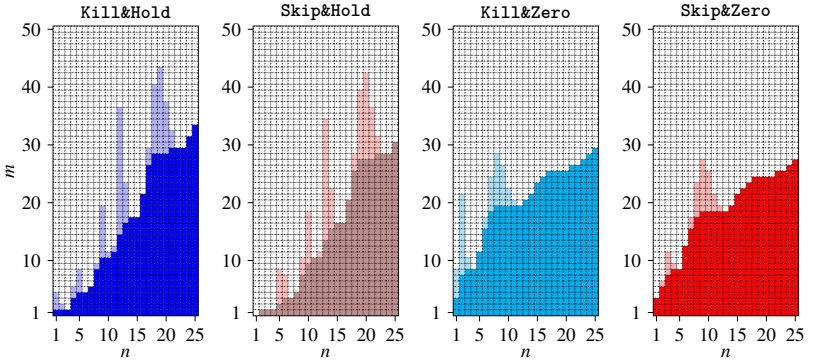


Figure 4. Miss-constrained stability (dark coloured area) and static-cyclic stability (light coloured area) when different strategies $\mathcal{H}$ are used in the example and the weakly hard model in Definition 2 is considered. Each square represents a window of size $\ell = m+n$. The dark area satisfies both the miss-constrained and static-cyclic stability whilst the light area only provides static-cyclic stability. The white squares denote potentially unstable combinations of m and n.

$\mathcal{P}$ is

$$\mathcal{P}\ :\ \begin{cases} x_{k+1} = \begin{bmatrix} 1.002 & 0.0100 & 0 & 0 \\ 0.3133 & 1.002 & 0 & 0 \\ -2.943\cdot 10^{-5} & -9.808\cdot 10^{-8} & 1 & 0.01 \\ -0.0059 & -2.943\cdot 10^{-5} & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} -0.0036 \\ -0.7127 \\ 0.0096 \\ 1.9120 \end{bmatrix} u_k + I w_k, \\ y_k = I x_k, \end{cases} \tag{24}$$

the controller $\mathcal{C}$ takes the form

$$\mathcal{C}\ :\ u_{k+1} = \begin{bmatrix} 8.8349 & 1.5804 & 0.2205 & 0.3049 \end{bmatrix} x_k \tag{25}$$

and is designed and analysed using the following parameters (see Section 3.3):

$$Q_e = \operatorname{diag}\{100,1,10,10\},\ Q_u = 100,\ R = \operatorname{diag}\{0,0,10,1\}. \tag{26}$$

We first apply the stability analyses presented in Section 5.2 to our model. Figure 4 shows the results. Each square in the figure represents a combination of (at most) m deadline misses (on the vertical axis) and (at least) n deadline hits (on the horizontal axis). If a square is coloured with a dark colour, the corresponding combination of misses and hits is both static-cyclic and miss-constrained stable, found using the `JSR Toolbox` [Vankeerberghen et al., 2014]. The light squares in the figure show combinations for which the system only satisfies the static-cyclic stability condition. The white squares mark configurations for which stability cannot be guaranteed.

We remark on the presence of peaks in the static-cyclic stability region of $\mathcal{H} = \texttt{KH}$ at $n = \{1,5,9,13,19\}$. Similar peaks are also found for the other strategies, but for different values of n. These peaks indicate that the system would be stable if that particular burst and recovery interval length would be repeated indefinitely. However, this assumption is not robust to variations in the burst or recovery interval lengths as can be seen from the miss-constrained stability region being more conservative with its guarantees. Instead, the peaks in the static-cyclic region can be explained by stable modes occurring due to the natural frequencies of the open-loop (for the `Zero` actuation mode) and closed-loop (for the `Hold` actuation mode) systems. It is also interesting to note that `Kill` seems to consistently yield a larger stability region than `Skip`, while neither `Zero` nor `Hold` dominate each other in terms of stability guarantees. An example of the latter fact was given already in [Schenato, 2009].

For the performance analysis, we considered a one-shot burst fault of a specific length m, followed by a long period of normal execution. Assuming that the pendulum starts close to the upright equilibrium, with stationary cost J_∞, we calculate how the covariance P_k and performance cost J_k evolve during and after the burst interval using Equations (20)–(21).[4] These calculations assume an ideal, linear model of the pendulum. The simulation results for different strategies and bursts of length $m = 20$ are shown in the upper half of Figure 5. For `Hold`, it is seen that the cost grows exponentially during the initial fault interval (the first $20T = 0.2\,\text{s}$). This is true also for `Zero`, although the growth rate is too small to be visible. The reason for the poor performance of `Hold` is that any non-zero held control signal will actively push the pendulum away from its unstable upright equilibrium even further than either disturbances or noise would already do without a proper control action.

The large spike in cost comes when the controller is reactivated at time $0.2\,\text{s}$. Here, the `Hold` strategy again shows much worse performance than `Zero`, with the peak cost being almost an order of magnitude worse. The difference between `Kill` and `Skip` is relatively small, with the latter strategy consistently performing slightly worse than the former. This is due to the small extra delay caused by using old data in the `Skip` strategy.

We conducted experiments on a Furuta pendulum, using the same controller for the real plant rather than its model.[5] Initially, we performed 500 experiments with 500 jobs each and no deadline misses, to determine the nominal variance of the system—i.e., the stationary variance used to find the static cost J_∞. For each

[4] The analysis is implemented using `JitterTime` [Cervin et al., 2019], `https://www.control.lth.se/jittertime`.

[5] A video, showing experiments with the real system and bursts of deadline misses can be viewed at `https://youtu.be/OPOK_7lvKVU`. The video shows a comparison of all the strategies for bursts of $(m = 20, n = 480)$. Furthermore, we have included additional experiments with $(m = 50, n = 450)$ and $(m = 75, n = 425)$ for the `Skip&Hold` strategy. The results of the additional experiments with higher values of m are not described in the paper, as stability could not be guaranteed (and in fact the pendulum is not at all times kept in the upright position).

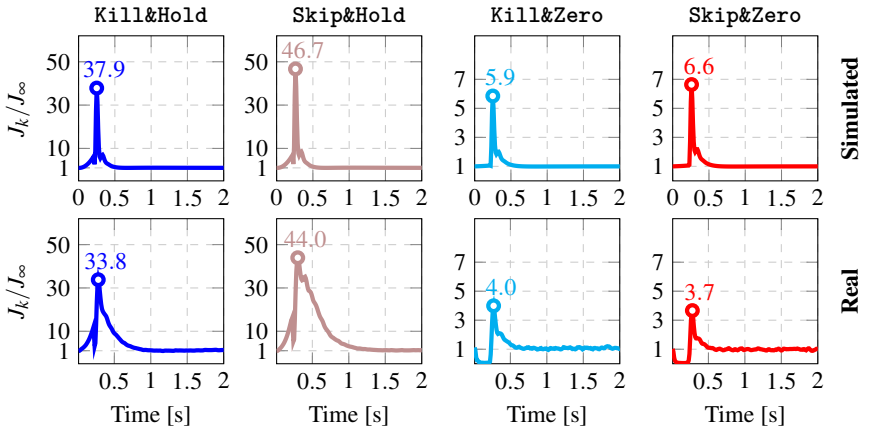


Figure 5. Normalised performance cost J_k/J_∞ obtained with the Furuta pendulum. The upper part of the figure shows simulated data, while the lower part of the figure shows the corresponding values obtained averaging the results of 500 experiments with the real process and hardware. Each experiment corresponds to a 500 jobs of the controller (20 misses and 480 hits).

strategy $\mathcal{H}$ we then ran 500 identically set up experiments. In each experiment, the control task operated according to the task model from Definition 2, experiencing a burst of length $m = 20$ misses, followed by by a recovery interval with $n = 480$ deadline hits.

Due to system model uncertainties (e.g., friction) being significant, the rotation angle around the arm axis displayed a considerable variance. We removed the state from the covariance calculations, since the arm angle majorly impacted the variance despite its inconsequential significance on the system dynamics (the pendulum can be stabilised with the arm being around any position, provided that the pendulum itself is kept in the upright position). Including the rotation angle would not change the shape of the performance degradation seen in Figure 5. However, it would make the results obtained with different strategies $\mathcal{H}$ not comparable (in some of them, the rotation angle could have varied less across the 500 experiments). The covariance matrix P_k was derived by calculating the variance of the closed-loop state vector $\tilde{x}_k$ according to Equation (9), in each time step k.

The resulting performance cost can be seen in the lower half of Figure 5, where the cost J_k was calculated according to Equation (11) and normalised using the stationary cost J_∞. Comparing the simulated (upper) and real (lower) performance costs in Figure 5, we notice the similarities between the simulated analysis and the analysis performed on the physical plant. Particularly, the strategies involving `Hold` actuation show similar behaviours. For these strategies, the simulated and real values are very close for the transient burst interval, the secondary cost peak (seen

around time 0.4 s), and the maximum normalised cost $J_{M,\mathcal{H}}$. However, the real cost is recovering slower than in the simulations—an effect that arises due to the nonlinear effects present in the real process, but unmodelled in the simulated environment. Instead, comparing the `Zero` actuation strategies, the performance cost of the physical experiments during the burst interval seem to improve compared to the simulations. This is again likely due to the unmodelled dynamics (e.g., friction) appearing in the physical experiment but not in the simulations. The stiction component of the friction reduces the variance of the states when the actuation signal becomes zero. With longer burst intervals, a similar behaviour as for the `Hold` actuation strategies would appear. Despite this difference, both the recovery interval, the secondary cost peak (around 0.4 s), and the maximum normalised costs $J_{M,\mathcal{H}}$ are comparable.

We conclude that the results of the experiments performed on the physical process support the validity of the performance analysis presented in Section 5.2.

6.2 Control Benchmark

In Section 6.1 we extensively discussed the results obtained with a single plant (the Furuta pendulum), with the aim of showing that simulating the performance cost yields interesting and relevant results. As the main novelty of this paper lays in the introduction of the performance analysis as an additional tool to evaluate the behaviour of control systems that can miss deadlines, we here focus on performance.

We use a set of representative process industrial plants [Åström and Hägglund, 2004], developed to benchmark PID design algorithms in the control literature. The set includes 9 different batches of stable plants, each presenting different features that can be encountered in process industrial plants, for a total of 133 plants.[6] For each batch, all systems have the same structure, but different parameters. For example, the fourth batch is a stable system with a set of repeated eigenvalues, and a single parameter specifying the system order, which can take six possible values (3, 4, 5, 6, 7, or 8). Almost all the plants have a single independent parameter. The only exception is Batch 7, for which we can specify two different configuration parameters, the first one having 4 possible values and the second one having 9 potential alternatives, with a total of 36 possible configurations.

The analysis methodology presented in this paper is valid for *all* linear control systems. In Section 6.1, we introduced an LQR controller to analyse the Furuta pendulum. To demonstrate the generality of the analysis, here, we focus on the most common controller class: proportional and integral (PI) controllers. These controllers constitute the vast majority of all the control loops in the process industry.[7] We also performed the analysis for proportional, integral, and derivative (PID) con-

[6] In our analysis, we present results with 134 plants. In fact, the test set was used in [Garpinger and Hägglund, 2015] to assess a control design method, and an additional plant was added to the set during this assessment. We included this additional plant in our analysis.

[7] A 2001 survey by Honeywell [Desborough, 2001] states that 97% of the existing industrial controllers are PI controllers.

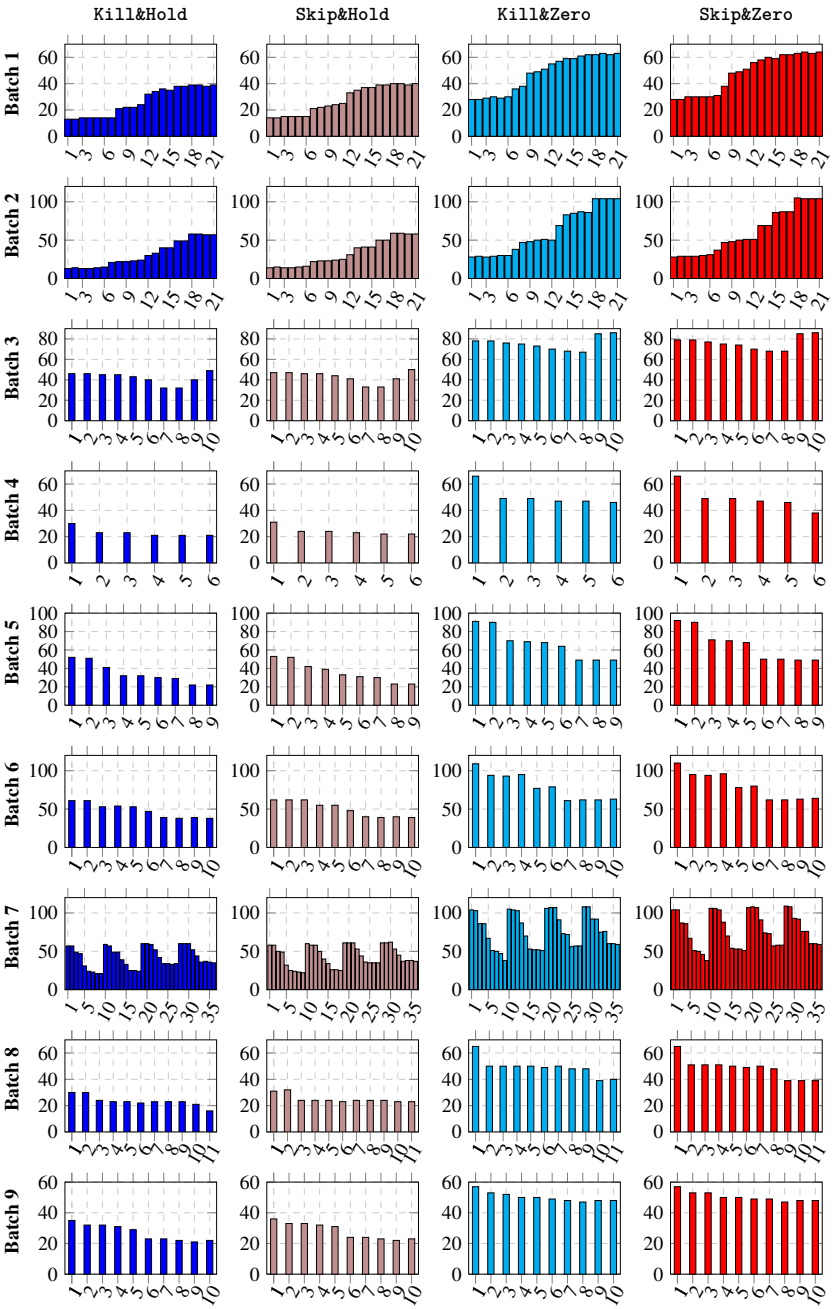


Figure 6. Performance Recovery Interval $n^*_{\mathcal{H}}$ needed to recover from a burst of 10 deadline misses for different strategies and all the plants in the 9 batches for PI controllers designed according to [Garpinger and Hägglund, 2015].

trollers obtaining similar results. Introducing our tuning for PID controllers requires additional clarifications and details, which we omit due to space limitations.

For each plant we derived a PI controller according to the methodology presented in [Garpinger and Hägglund, 2015]. In order to showcase the applicability of our analysis to different linear systems, controllers, and noise models, we analyse the resulting closed-loop systems for $m \in [1, 20]$, under the assumption that the systems are affected by brown noise (in comparison to the white noise applied to the Furuta Pendulum). The brown noise model integrates the white noise and is thus applicable to systems where the noise is more dominant at lower frequencies (e.g., oscillations from nearby machinery). Figure 6 shows the results for $m = 10$.

The first result that the figure shows is that the plant dynamics plays an important role in how the system reacts to misses. For example, the plants in Batch 4 and Batch 8 need around 20 hits to recover from a burst of 10 misses. On the contrary, the plants in Batch 6 and Batch 7 need a higher number of hits to recover from the same burst interval. The second result that is apparent from the figure is that the `Hold` actuation strategy recovers much better (performance-wise) than `Zero`. The reason why `Hold` outperforms `Zero` can be explained by the brown noise. The control signal will actively counteract the integrated noise dynamics, meaning that zeroing the control signal removes the compensation against the integrated noise. Finally, comparing the deadline handling strategies, `Kill` performs marginally better than `Skip`. Under `Kill`, the controller uses fresh data at the beginning of the recovery interval, while `Skip` uses old data. However, we assumed ideal rollback (i.e., zero additional computation time for the rollback and clean state) for the `Kill` strategy. In real systems, rollback is difficult to realise and the advantage provided by `Kill` over `Skip` may therefore become unimportant. These findings are consistent throughout all the plants in the experimental set, regardless of the burst interval length m.

The plant dynamics and noise affect the behaviour and performance of the strategies. Comparing the results of Section 6.1 with the aggregate results, it becomes apparent that the actuation strategy (`Zero` or `Hold`) affects control performance significantly more than the deadline handling strategy. For the Furuta pendulum (an unstable, nonlinear plant influenced by white noise) `Zero` performed the best, but for the process industrial systems (stable, linear plants influenced by brown noise) `Hold` outperformed `Zero`. These results were apparent even with no consideration taken to the deadline handling strategies. Thus, we conclude that the plant and noise model should be the ruling factor when choosing the actuation strategy, while the deadline handling strategy is mainly limited by the constraints imposed by the real-time implementation.

7. Conclusions

In this paper we analysed control systems and their behaviour in the presence of bursts of deadline misses. We provided a comprehensive set of tools to determine how robust a given control system is to faults that hinder the computation to complete in time, with different handling strategies. Our analysis tackles both stability and performance. In fact, we have shown that analysing the stability of the system is not enough to properly quantify the robustness to deadline misses, as the performance loss could be significant even for stable systems. We introduced two performance metrics, linked to the recovery of a system from a burst of deadline misses.

A limitation of the presented performance analysis is that it only applies to linear control systems. However, the approach can easily be extended to analyse *time-varying* linear systems and can also be used for local analysis of a nonlinear system that should follow a given reference trajectory. In fact, to illustrate the applicability to real (e.g., nonlinear) systems, we applied the analysis to a Furuta pendulum and compared the results of simulations obtained with a model of the process to the real execution data. The results support our claim that the proposed performance analysis is a valid approximation of the real-world system performance.

We performed additional tests on a large batch of industrial plants, using modern control design techniques. From our experimental campaign, we conclude that the choice of actuation strategy affects the control performance significantly more than the choice of deadline handling strategy.

Acknowledgements

The authors are members of the ELLIIT Strategic Research Area at Lund University. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Number 871259 (ADMORPH project). This publication reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

References

Abdi, F., C. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo (2019). "Preserving physical safety under cyber attacks". *IEEE Internet of Things Journal* **6**:4.

Abdi, F., R. Mancuso, R. Tabish, and M. Caccamo (2017a). "Restart-based fault-tolerance: system design and schedulability analysis". In: *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*.

Abdi, F., R. Tabish, M. Rungger, M. Zamani, and M. Caccamo (2017b). "Application and system-level software fault tolerance through full system restarts". In: *8th International Conference on Cyber-Physical Systems (ICCPS)*. ISBN: 9781450349659.

Ahrendts, L., S. Quinton, T. Boroske, and R. Ernst (2018). "Verifying weakly-hard real-time properties of traffic streams in switched networks". In: *30th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 15:1–15:22. ISBN: 978-3-95977-075-0.

Åkesson, B., M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis (2020). "An empirical survey-based study into industry practice in real-time systems". In: *41st IEEE Real-Time Systems Symposium (RTSS)*.

Altmeyer, S. and R. I. Davis (2014). "On the correctness, optimality and precision of static probabilistic timing analysis". In: *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6.

Åström, K. J. and T. Hägglund (2004). "Revisiting the Ziegler-Nichols step response method for PID control". *Journal of Process Control* **14**:6, pp. 635–650. ISSN: 0959-1524.

Åström, K. J. and B. Wittenmark (1997). *Computer-Controlled Systems (3rd Ed.)* Prentice-Hall, Inc., USA. ISBN: 0133148998.

Åström, K. and T. Hägglund (2006). *Advanced PID Control*. English. ISA - The Instrumentation, Systems and Automation Society. ISBN: 978-1-55617-942-6.

Bernat, G. and A. Burns (1997). "Combining $\binom{n}{m}$-hard deadlines and dual priority scheduling". In: *18th IEEE Real-Time Systems Symposium (RTSS)*, pp. 46–57.

Bernat, G., A. Burns, and A. Liamosi (2001). "Weakly hard real-time systems". *IEEE Transactions on Computers* **50**:4, pp. 308–321. DOI: 10.1109/12.919277.

Bund, T. and F. Slomka (2014). "Controller/platform co-design of networked control systems based on density functions". In: *4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*. ACM, pp. 11–14. ISBN: 978-1-4503-2871-5.

Bund, T. and F. Slomka (2015). "Worst-case performance validation of safety-critical control systems with dropped samples". In: *23rd International Conference on Real Time and Networks Systems (RTNS)*. ACM, Lille, France, pp. 319–326. ISBN: 978-1-4503-3591-1.

Buttazzo, G., M. Velasco, and P. Marti (2007). "Quality-of-control management in overloaded real-time systems". *IEEE Transactions on Computers* **56**:2, pp. 253–266.

Caccamo, M. and G. Buttazzo (1997). "Exploiting skips in periodic tasks for enhancing aperiodic responsiveness". In: *18th IEEE Real-Time Systems Symposium (RTSS)*, pp. 330–339.

Caccamo, M., G. Buttazzo, and L. Sha (2000). "Capacity sharing for overrun control". In: *21st IEEE Real-Time Systems Symposium (RTSS)*, pp. 295–304.

Cazzolato, B. S. and Z. Prime (2011). "On the dynamics of the Furuta pendulum". *Journal of Control Science and Engineering*.

Cervin, A., P. Pazzaglia, M. Barzegaran, and R. Mahfouzi (2019). "Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems". In: *IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1025–1032.

Cervin, A. (2005). "Analysis of overrun strategies in periodic control tasks". *IFAC Proceedings Volumes* **38**:1. 16th IFAC World Congress, pp. 219–224. ISSN: 1474-6670. DOI: 10.3182/20050703-6-CZ-1902.01076.

Chen, A., H. Xiao, A. Haeberlen, and L. T. X. Phan (2015). "Fault tolerance and the five-second rule". In: *Workshop on Hot Topics in Operating Systems (HotOS)*.

Choi, H., H. Kim, and Q. Zhu (2019). "Job-class-level fixed priority scheduling of weakly-hard real-time systems". In: *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 241–253.

Davis, R. I., L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean (2013). "Analysis of probabilistic cache related pre-emption delays". In: *25th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 168–179.

de Niz, D., L. Wrage, A. Rowe, and R. Rajkumar (2013). "Utility-based resource overbooking for cyber-physical systems". In: *19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 217–226.

Desborough, L. (2001). "Increasing customer value of industrial control performance monitoring-honeywell's experience". *Preprints of CPC*, pp. 153–186.

Ernst, R., S. Kuntz, S. Quinton, and M. Simons (2018). "The logical execution time paradigm: new perspectives for multicore systems". *Dagstuhl Reports* **8**, pp. 122–149.

Frehse, G., A. Hamann, S. Quinton, and M. Woehrle (2014). "Formal analysis of timing effects on closed-loop properties of control software". In: *35th IEEE Real-Time Systems Symposium (RTSS)*, pp. 53–62.

Furuta, K., M. Yamakita, and S. Kobayashi (1992). "Swing-up control of inverted pendulum using pseudo-state feedback". *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* **206**:4, pp. 263–269.

Garpinger, O. and T. Hägglund (2015). "Software-based optimal PID design with robustness and noise sensitivity constraints". *Journal of Process Control* **33**, pp. 90–101. ISSN: 0959-1524.

Gaukler, M., T. Rheinfels, P. Ulbrich, and G. Roppenecker (2019). "Convergence rate abstractions for weakly-hard real-time control". *arXiv preprint arXiv:1912.09871*.

Ghosh, S. K., S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty (2018). "Design and validation of fault-tolerant embedded controllers". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*. IEEE.

Goswami, D., D. Mueller-Gritschneder, T. Basten, U. Schlichtmann, and S. Chakraborty (2014). "Fault-tolerant embedded control systems for unreliable hardware". In: *International Symposium on Integrated Circuits (ISIC)*. IEEE.

Gujarati, A., M. Nasri, and B. B. Brandenburg (2018). "Quantifying the resiliency of fail-operational real-time networked control systems". In: *30th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-075-0.

Gujarati, A., M. Nasri, R. Majumdar, and B. B. Brandenburg (2019). "From iteration to system failure: characterizing the fitness of periodic weakly-hard systems". In: *31st Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Hamdaoui, M. and P. Ramanathan (1995). "A dynamic priority assignment technique for streams with (m,k)-firm deadlines". *IEEE Transactions on Computers* **44**:12, pp. 1443–1451.

Hammadeh, Z. A. H., R. Ernst, S. Quinton, R. Henia, and L. Rioux (2017a). "Bounding deadline misses in weakly-hard real-time systems with task dependencies". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*, pp. 584–589.

Hammadeh, Z. A. H., S. Quinton, and R. Ernst (2014). "Extending typical worst-case analysis using response-time dependencies to bound deadline misses". In: *14th International Conference on Embedded Software (EMSOFT)*. ACM. ISBN: 9781450330527.

Hammadeh, Z. A. H., S. Quinton, and R. Ernst (2019). "Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks". *ACM Transactions of Embedded Computing Systems* **18**:6. ISSN: 1539-9087.

Hammadeh, Z. A. H., S. Quinton, M. Panunzio, R. Henia, L. Rioux, and R. Ernst (2017b). "Budgeting under-specified tasks for weakly-hard real-time systems". In: *29th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 76. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 17:1–17:22. ISBN: 978-3-95977-037-8.

Hertneck, M., S. Linsenmayer, and F. Allgöwer (2019). "Nonlinear dynamic periodic event-triggered control with robustness to packet loss based on non-monotonic lyapunov functions". In: *58th IEEE Conference on Decision and Control (CDC)*, pp. 1680–1685.

Jungers, R. (2009). *The Joint Spectral Radius: Theory and Applications*. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg. ISBN: 9783540959809.

Kauer, M., D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy (2014). "Fault-tolerant control synthesis and verification of distributed embedded systems". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*.

Khosravi, F., M. GlaSS, and J. Teich (2017). "Automatic reliability analysis in the presence of probabilistic common cause failures". *IEEE Transactions on Reliability* **66**:2.

Khosravi, F., M. Müller, M. GlaSS, and J. Teich (2015). "Uncertainty-aware reliability analysis and optimization". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 97–102. ISBN: 9783981537048.

Kirsch, C. and A. Sokolova (2012). "The logical execution time paradigm". In: *Advances in Real-Time Systems*. Springer Berlin Heidelberg, pp. 103–120. ISBN: 978-3-642-24349-3.

Koren, G. and D. Shasha (1995). "Skip-Over: algorithms and complexity for overloaded systems that allow skips". In: *16th IEEE Real-Time Systems Symposium (RTSS)*, pp. 110–117.

Linsenmayer, S. and F. Allgower (2017). "Stabilization of networked control systems with weakly hard real-time dropout description". In: *56th IEEE Conference on Decision and Control (CDC)*, pp. 4765–4770.

Linsenmayer, S., M. Hertneck, and F. Allgower (2020). "Linear weakly hard real-time control systems: time- and event-triggered stabilization". *IEEE Transactions on Automatic Control*.

Maggio, M., A. Hamann, E. Mayer-John, and D. Ziegenbein (2020). "Control-system stability under consecutive deadline misses constraints". In: *32nd Euromicro Conference on Real-Time Systems (ECRTS)*. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Montgomery, D. (2009). *Introduction to Statistical Quality Control*. Wiley. ISBN: 9780470233979.

Natarajan, S., M. Nasri, D. Broman, B. B. Brandenburg, and G. Nelissen (2019). "From code to weakly hard constraints: a pragmatic end-to-end toolchain for timed C". In: *40th IEEE Real-Time Systems Symposium (RTSS)*, pp. 167–180.

OConnor, P. P. and A. Kleyner (2012). *Practical Reliability Engineering*. 5th. Wiley Publishing. ISBN: 047097981X.

Palopoli, L., L. Abeni, G. Buttazzo, F. Conticelli, and M. Di Natale (2000). "Real-time control system analysis: an integrated approach". In: *21st IEEE Real-Time Systems Symposium (RTSS)*, pp. 131–140.

Pazzaglia, P., A. Hamann, D. Ziegenbein, and M. Maggio (2021). "Adaptive design of real-time control systems subject to sporadic overruns". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*.

Pazzaglia, P., C. Mandrioli, M. Maggio, and A. Cervin (2019). "DMAC: Deadline-Miss-Aware Control". In: *31st Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 1:1–1:24. ISBN: 978-3-95977-110-8.

Pazzaglia, P., L. Pannocchi, A. Biondi, and M. D. Natale (2018). "Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses". In: Altmeyer, S. (Ed.). *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:22. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.10.

Quinton, S., T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst (2014). "Typical worst case response-time analysis and its use in automotive network design". In: *51st Annual Design Automation Conference (DAC)*. ACM, San Francisco, CA, USA, pp. 1–6. ISBN: 9781450327305.

Ramanathan, P. (1997). "Graceful degradation in real-time control applications using (m,k)-firm guarantee". In: *27th IEEE International Symposium on Fault Tolerant Computing*, pp. 132–141.

Schenato, L. (2009). "To zero or to hold control inputs with lossy links?" *IEEE Transactions on Automatic Control* **54**:5, pp. 1093–1099.

Soudbakhsh, D., L. T. X. Phan, O. Sokolsky, I. Lee, and A. Annaswamy (2013). "Co-design of control and platform with dropped signals". In: *4th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. ACM, pp. 129–140. ISBN: 978-1-4503-1996-6.

Soudbakhsh, D., L. T. X. Phan, A. M. Annaswamy, and O. Sokolsky (2018). "Co-design of arbitrated network control systems with overrun strategies". *IEEE Transactions on Control of Network Systems* **5**:1, pp. 128–141. DOI: 10.1109/TCNS.2016.2583064.

Sun, Y. and M. D. Natale (2017). "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks". *ACM Transactions on Embedded Computing Systems* **16**:5s. ISSN: 1539-9087.

Vankeerberghen, G., J. Hendrickx, and R. M. Jungers (2014). "JSR: a toolbox to compute the joint spectral radius". In: *17th International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, Berlin, Germany, pp. 151–156. ISBN: 9781450327329.

Vreman, N. and C. Mandrioli (2020). "Evaluation of Burst Failure Robustness of Control Systems in the Fog". In: *2nd Workshop on Fog Computing and the IoT (Fog-IoT)*. Vol. 80. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-95977-144-3. DOI: 10.4230/OASIcs.Fog-IoT.2020.8.

Xu, W., Z. A. H. Hammadeh, A. Kröller, R. Ernst, and S. Quinton (2015). "Improved deadline miss models for real-time systems using typical worst-case analysis". In: *27th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 247–256.

Paper II

Deadline-Miss-Adaptive Controller Implementation for Real-Time Control Systems

Nils Vreman Claudio Mandrioli Anton Cervin

Abstract

The policy used to implement a control algorithm in a real-time system can significantly affect the quality of control. In this paper, we present a method to adapt the controller implementation, with the objective to improve the system's performance under real-time faults. Our method compensates for missing state updates by adapting the controller parameters according to the number of consecutively missed deadlines. It extends the state-of-the-art by considering dynamic controllers, which have had limited coverage in previous literature. The adaptation mechanism can be precomputed offline, solely based on knowledge about the controller and not on the controlled plant. The approach is indifferent to the control design, as well as to the scheduling policy, and can be automatically realised by the operating system, thus improving the robustness of the control system to intermittent and unexpected real-time faults. We develop a stochastic performance analysis method and apply it to both a real plant and numerous simulated plants to evaluate our adaptive controller. Complementary to the stochastic analysis, we also do worst-case stability analysis of the resulting system. The results confirm the conjuncture that the adaptive controller improves both the performance and robustness in the presence of deadline misses.

Originally published in IEEE 28th Real-Time and Embedded Technology and Applications Symposium (2022). The mathematical notation has been unified to match the remainder of the thesis. Reprinted with permission.

1. Introduction

Computer-controlled systems are prime instances of real-time systems [Oshana, 2006; Akesson et al., 2020]. Due to the tight interconnection between the environment, hardware, and software, designing such systems has been considered challenging. Part of this challenge resides in the design of the real-time software, specifically considering both the normal operation [Lozoya et al., 2013; Aminifar et al., 2011] (e.g., correct output computation) and system malfunctions [Caccamo et al., 2002; Ramanathan, 1997] (e.g., temporary overloads). For this reason, the research community has undertaken a significant effort to merge design choices on the algorithmic side and on the real-time implementation side.

In computer-controlled systems, algorithms are developed using control theory [Åstrom and Murray, 2008]. The theory offers strong and practically relevant formal guarantees, but also makes strict assumptions on the real-time execution of the implemented algorithm. These assumptions are naturally translated into periodic tasks with hard deadlines. However, meeting every single deadline in a periodic control task is not necessary [Ramamritham, 1996; Ramanathan, 1997]. Instead, the timing requirements are the result of design choices and engineering trade-offs between resource utilisation and performance [Lozoya et al., 2013; Cervin et al., 2004].

Co-design approaches have been proposed to maximise the control performance while minimising the real-time resource utilisation [Marti et al., 2001; Rehbinder and Sanfridson, 2000]. However, the tight integration between control algorithms and the real-time implementation has limitations, due to the complexity of the resulting systems. This complexity generally translates into (i) complex design methodologies, (ii) conservative results, and (iii) strong assumptions on the system properties.

Differently from existing approaches, in this paper we avoid the additional complexity by *automatically* adapting the real-time implementation of the controller. The approach is inspired by the concept of autotuning, originating in the control literature [Åström and Hägglund, 1984; Hägglund and Åström, 1983]. Autotuning was developed to simplify the PID control design process by automatically optimising the controller's design parameters. This idea is here translated to the real-time implementation, where instead the predesigned control algorithm is automatically adapted for the real-time architecture to minimise the design effort.

To enable an automatic adaptation of the control algorithm for a wide range of systems, we make as few assumptions about the implementation platform as possible. The system requirements that we pose are nonintrusive and seen in industrial applications [Akesson et al., 2020]. The real-time operating system is required to be able to: schedule periodic tasks with implicit deadlines, abort (`Kill`) instances of tasks that miss their deadlines, roll back the state of aborted tasks [Ying Zhang and Krishnendu Chakrabarty, 2003; Seong Woo Kwak et al., 2001], and handle controller inputs and outputs at task release times [Kirsch and Sokolova, 2012; Ernst

et al., 2018]. While previous co-design approaches require, e.g., probabilistic or weakly hard descriptions of deadline misses [Pazzaglia et al., 2019; Kauer et al., 2014], our adaptation approach works for any deadline miss model. From the point of view of the controller, we assume a linear time-invariant control law, but we require no prior information about the control design, nor about the system to be controlled. Similarly to [Pazzaglia et al., 2021], our adaptive control implementation is applicable to general linear discrete-time controllers and not only static ones.

To evaluate the performance of our adaptive implementation, we propose a stochastic analysis of the control system. The analysis assumes a probabilistic model of the deadline misses; however, it is agnostic to how the model is obtained. We utilise this analysis to evaluate our approach on both a *real* system and a benchmark set of 268 *simulated* control systems from the process industrial domain. We use the former to evaluate the practical relevance of the proposed approach and the latter to evaluate its general applicability. We complement our performance analysis with a worst-case stability analysis. In all of our tests, the adaptive implementation improves both the performance and the worst-case stability of the system.

The paper provides the following two main contributions:

- It proposes a novel, modular and intuitive control law implementation that adapts the control action upon the occurrence of deadline misses in the periodic controller task. The adaptive implementation has a small overhead and is applicable to all linear dynamic controllers.

- It proposes a probabilistic analysis of the resulting control system subject to deadline misses. The analysis is based on a comparison with the ideal system without deadline misses and is used to evaluate the performance of both a real system and numerous simulated control systems. The results show that the adaptive implementation significantly improves the system performance and robustness, compared to the nominal implementation.

The remainder of this paper is outlined as follows. In Section 2 we present the relevant control and real-time system background. Section 3 presents and discusses the previous literature on the topic, identifying the limitations of the state-of-the-art that we are addressing. In Section 4 we first propose our adaptive implementation of the control law, and then we integrate the proposed controller with a probabilistic analysis method to enable its evaluation. Section 5 presents an empirical evaluation of the adaptive control law based on the proposed analysis for both a physical system and numerous simulated systems. Finally, Section 6 concludes the paper.

2. System Model

This section introduces the necessary background and models needed for the remainder of the paper. We discuss the real-time implementation of a general linear

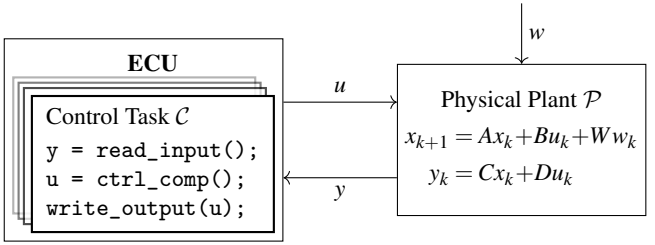


Figure 1. Typical structure of a computer-controlled system. *Left:* the Electronic Control Unit (ECU) implementing a real-time system, including a task executing the controller. *Right:* the physical plant controlled by the actuation variable u, affected by the disturbance w, and producing the measurement y.

controller and how it can affect the performance of the system. We start by describing the behaviour of the system under *ideal* conditions. Based on this, we state how *deadline misses* in the real-time implementation affect the system's behaviour.

2.1 Control Systems under Ideal Operations

The objective of a control system is to regulate a physical process, usually called a *plant*, so that it behaves as desired. Figure 1 shows the structure of a control system, where an Electronic Control Unit (ECU) implements a real-time system. Among the different tasks executed in the system, there is a task responsible for the control computations, denoted as the control task. Every job released by this task performs the following actions: (i) Read measurements from the sensors. (ii) Use the sensor information to update its state and compute a control action. (iii) Write the control action to the actuators. The executed algorithm is generally designed using control theory, where the effective application of the algorithm relies on assumptions about the real-time execution.

The dynamic behaviour of a plant is commonly described using a state-space model [Åstrom and Murray, 2008]. Such models are constituted of two sets of equations: one describing the dynamics of the plant and another one describing the relation between the plant state and the available measurements. Describing physical phenomena, those equations are for the most part continuous and nonlinear. However, for control design and implementation purposes, they are commonly transformed into a discrete-time linear time-invariant (LTI) state-space model:

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

Here, the variable k counts the number of discrete time steps that have passed since the system started executing. Furthermore, the variable $x_k \in \mathbb{R}^{n_x}$ represents the *plant state*, $u_k \in \mathbb{R}^{n_u}$ corresponds to the *control signal* computed in order to affect the plant, $w_k \in \mathbb{R}^{n_w}$ models disturbances and reference signals, and $y_k \in \mathbb{R}^{n_y}$ is

the *measurement signal* available to the controller. For what concerns the matrices, $A \in \mathbb{R}^{n_x \times n_x}$ captures the relation between the current state and the next state, while $B \in \mathbb{R}^{n_x \times n_u}$ and $W \in \mathbb{R}^{n_x \times n_w}$ respectively capture how the control signal and the exogenous signals affect the state at the next time step. Furthermore, $C \in \mathbb{R}^{n_y \times n_x}$ and $D \in \mathbb{R}^{n_y \times n_u}$ respectively describe how the current state and control signal relate to the measurements.

To control the behaviour of the plant, a *controller* $\mathcal{C}$ is synthesised to follow some desired properties, such as: (i) stability, (ii) speed of convergence, (iii) control effort, and (iv) disturbance rejection. The stability requirement enforces that none of the signals diverge and is a necessary condition for all controllers. Moreover, a controller that fulfils all requirements will make the output converge to the reference value within a specified time, while minimising the control effort and the effect of possible disturbances.

Controllers are commonly implemented as fixed-rate, periodically executing tasks, following the Logical Execution Time (LET) paradigm [Henzinger et al., 2003; Kirsch and Sokolova, 2012; Ernst et al., 2018]. Adopting this paradigm, the sensors are read at the beginning of the task period, and the control action is written to the actuators at the end of the period. This minimises the effect of fluctuations in the execution pattern of the control algorithm (called jitter) at the cost of introducing a one-step delay in the actuation.

While they are sometimes specified as transfer functions [Åstrom and Murray, 2008], controllers are often *implemented* as discrete-time state-space systems. More specifically, we assume that the controller is an LTI system that takes the measurement y_k as input and produces the control signal u_{k+1} as output:[1]

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

Here, $z_k \in \mathbb{R}^{n_z}$ represents the internal state of the controller. The second equation, specifying the control action at step $k+1$, captures the one-step delay introduced by the LET paradigm. The matrices $F \in \mathbb{R}^{n_z \times n_z}$, $G \in \mathbb{R}^{n_z \times n_y}$, $H \in \mathbb{R}^{n_u \times n_z}$, and $K \in \mathbb{R}^{n_u \times n_y}$ govern the behaviour of the controller.

In conjunction with Equation (2), we define two types of controllers: *static* and *dynamic*.

DEFINITION 1—STATIC CONTROLLER
We denote a static controller *as any controller $\mathcal{C}$ that is* stateless *(i.e., it has no internal state z; $n_z = 0$).*

DEFINITION 2—DYNAMIC CONTROLLER
We denote a dynamic controller *as any controller $\mathcal{C}$ that is* stateful *(i.e., it has an internal state z; $n_z \geq 1$).*

[1] Note that we adopt a positive feedback convention.

From the definitions above, we note that a static controller can be written as a fixed gain matrix times the input (i.e., $\mathcal{C} : u_{k+1} = Ky_k$), while a dynamic controller is equivalent to (2) with non-empty matrices F, G, H, and K. Examples of static controllers include proportional (P) controllers, state feedback controllers, and linear–quadratic regulators (LQR), while dynamic controllers include proportional–integral–derivative (PID) controllers, lead–lag compensators, and linear–quadratic–Gaussian (LQG) regulators [Åstrom and Murray, 2008].

2.2 Control Systems Subject to Deadline Misses

We assume that the controller $\mathcal{C}$ is implemented as a periodic task with period T and implicit deadlines. Intuitively, each execution period of the control task corresponds to one time step k. At the start of period k, the task releases a *job* that should be completed before the deadline at time $(k+1)T$.

Faults in the real-time system can affect the timely execution of the control task [Steinbauer, 2013]. We denote the outcome of a job's execution as either a deadline *hit* or *miss*, corresponding to whether the job completed its execution before its deadline or not. The source of a deadline miss could be a temporary CPU overload [Baruah and Haritsa, 1997], cache misses [Milligan and Cragon, 1996; Wang et al., 2012], or unexpected preemption from hardware interrupts or higher priority tasks [Stankovic et al., 1995]. However, the analysis and adaptation methods presented in this paper are independent of the origin of the deadline miss.

To study what happens when the controller misses a deadline, we have to define how the system behaves in such circumstances. In particular, three aspects need to be considered: (i) how the controller state is updated, (ii) how the actuator handles the lack of a new control signal [Schenato, 2009], and (iii) how the operating system handles a job that misses its deadline [Pazzaglia et al., 2019; Cervin, 2005]. The first item refers to what happens to the internal state z when the controller is unable to finish its execution ahead of its dedicated deadline. Henceforth, we assume that when the controller misses its deadline, the controller state is *not* updated (implying $z_{k+1} = z_k$). This is motivated by the possibility to roll back z_k to a previous state [Akesson et al., 2020; Seong Woo Kwak et al., 2001; Ying Zhang and Krishnendu Chakrabarty, 2003] and by the impossibility to guarantee that the state update was finished if the job was only partially completed.

Regarding the second item, mainly two actuator models have previously been considered in the literature: `Zero` and `Hold` [Schenato, 2009]. Under the `Zero` model, if the job released at time kT misses its deadline at time $(k+1)T$, the actuator outputs $u_{k+1} = 0$. This strategy is uncommon in practice, because it performs well only in very specific cases [Vreman et al., 2021a]. Instead, the more common actuator model is to hold the control signal in the case of a missed deadline: $u_{k+1} = u_k$. Although our proposed adaptation is in itself independent of the actuator model, the `Hold` model has been adopted in the analysis and examples of this paper.

For what concerns the handling of the job that missed the deadline, there exist

at least three different employable strategies: (a) `Kill` (b) `Skip`, and (c) `Queue`. When using the `Kill` strategy, the job that missed its deadline gets terminated, the controller state is rolled back, and the next job is released. The `Skip` strategy does not terminate the job that missed its deadline. Instead, it lets the job continue its execution, not releasing subsequent jobs until the active one has finished executing. `Queue` behaves similarly to the `Skip` strategy: it does not kill the current job, but it does release the subsequent jobs to the job queue. Both the `Skip` and `Queue` strategies allow jobs to work with outdated input data, since they do not terminate jobs that miss their deadline. Thus, they introduce a lag in the actuation of the control law, which in most cases reduces the control performance with respect to what could be achieved with the `Kill` strategy.

For the remainder of this paper we will use the `Kill` strategy, since it (a) introduces explicit breakpoints in which to adapt the controller, (b) supplies the controller with the latest sensor measurement after an overrun, (c) helps free computational resources in overrun situations, and (d) is a common choice in both industry and literature [Akesson et al., 2020; Bernat et al., 2001; Hertneck et al., 2019]. Furthermore, we note that in [Vreman et al., 2021a] it has been observed that the actuator model is of greater relevance than the handling of the deadline overrun. We also note that an effective implementation of the `Kill` strategy that includes state roll-back requires the implementation of checkpointing mechanisms [Ying Zhang and Krishnendu Chakrabarty, 2003; Seong Woo Kwak et al., 2001], which might not be implemented by default in a real-time system.

We conclude this section by mentioning that various models have been proposed to describe tasks that may experience deadline misses, e.g., soft [Marchand and Chetto, 2008] or weakly hard models [Bernat et al., 2001; Hammadeh et al., 2017]. For the adaptive controller in this work, we only assume that the number of consecutive deadline misses q is bounded by a finite quantity $q_{max} < \infty$. This assumption is a practical necessity, since accepting an infinite run of deadline misses would completely disconnect the plant from the controller [Maggio et al., 2020].

Solely for performance analysis, in Section IV-B we adopt a stochastic model of the sequences of deadline hits and misses. Such a model can be computed from a probabilistic task set model using existing techniques [Chen and Chen, 2017; Chen et al., 2018; Markovi et al., 2021]. However, we remark that the proposed adaptive controller implementation is independent of the stochastic deadline miss model.

2.3 Control System Stability under Deadline Misses

Guaranteeing the stability of control systems is essential in control engineering. A linear control system without exogenous inputs is *stable* if and only if the system's state always converges to zero irrespective of its initial value. Assuming ideal conditions, i.e., no deadline misses, stability can be verified by checking whether all the closed-loop system's eigenvalues lie inside the unit circle. However, in the presence of deadline misses, the classical stability criteria are no longer sufficient,

since the dynamics of the control system is time-varying and changes according to the specific pattern of deadline misses. For such cases, switched system stability analysis (also known as switching stability) is a viable extension of classical stability [Liberzon, 2003]. In this paper, we analyse the switching stability using both a time-averaged (Markov Jump Linear Analysis [Fang and Loparo, 2002]) and a worst-case (Joint Spectral Radius [Rota and Strang, 1960]) approach.

Modelling the deadline misses as a stationary random process with known statistical properties allows us to calculate an analytical time-averaged performance index of the closed-loop system. If the performance index is finite, then the system is guaranteed to be stable in the mean-square sense (meaning that the state will not diverge with probability 1). We develop our approach to compute the time-averaged performance in Section 4.2.

If the statistical properties of the deadline misses are unknown or uncertain, switching stability can still be analysed under worst-case conditions. The Joint Spectral Radius (JSR) generalises the spectral radius of a matrix (i.e., the largest absolute eigenvalue) to a set of matrices; thus, the JSR characterises the largest asymptotic growth (or contraction) rate of the states. If the JSR of the set of closed-loop matrices representing $i = \{0, 1, \ldots, q_{max}\}$ deadline misses (followed by a hit) is below 1 then the system is switching stable. Conversely, if it is above 1, there exists at least one sequence of deadline misses and hits that makes the system unstable. There exist both toolboxes and methods for calculating the JSR for control systems subject to deadline misses [Vankeerberghen et al., 2014; Maggio et al., 2020; Vreman et al., 2021b].

3. Problem Description

In this section, we discuss the problem of implementing a control system, as defined in Section 2, that is robust to deadline misses. The problem has been investigated in the existing literature, and different methods have been proposed to solve it. We offer a discussion on the scientific literature to identify the strengths and limitations of previous work and motivate the research problem studied in this paper.

3.1 Related Work

The robustness of control algorithms to timing faults has been discussed both from the analysis and synthesis perspectives. From the *analysis* perspective, the literature proposes different methods to analyse the stability and performance of control systems in the presence of real-time faults. The performance of different overrun handling strategies was discussed in the context of control systems in [Cervin, 2005]. Similar studies, in the context of networked control systems, were presented by [Schenato et al., 2007] and [Vreman and Mandrioli, 2020]. In [Frehse et al., 2014], the authors developed a network of hybrid automata to analyse the consequences of timing variations in control software. The analysis of control systems un-

der consecutive deadline misses was addressed in [Maggio et al., 2020] and [Xiong and Lam, 2007], using respectively the joint spectral radius and Lyapunov theory. Both [Hertneck et al., 2021] and [Hertneck et al., 2020] also leveraged Lyapunov theory to analyse nonlinear systems subject to network problems. Shifting the focus from stability to performance, [Vreman et al., 2021a] analysed control systems subject to bursts of deadline misses.

From the *synthesis* perspective, many works have studied overruns from a control and scheduling co-design perspective [Årzén et al., 2000]. In [Schinkel and Chen, 2006], the authors designed state feedback controllers with time-varying gain, guaranteeing control performance under arbitrary period changes of the control task. A similar approach was taken in [Ramanathan, 1997], where instead a weakly hard task model with known execution pattern was assumed. The authors of [Kumar et al., 2012] designed stabilising state feedback controllers with different controller gains depending on the system's time delay. In [Pazzaglia et al., 2019], the authors used a probabilistic characterisation of the task model to develop state feedback controllers that are robust to deadline overruns. Motivated by industrial practices, [Pazzaglia et al., 2021] proposed the re-initialisation of the control task's period after the occurrence of overruns as well as the design of an adaptive control law. In the field of networked control [Gupta and Chow, 2010; Törngren, 1998], different works have proposed compensation schemes for jitter and dropped data packets [Nilsson et al., 1998; Zhang et al., 2001; Hespanha et al., 2007]. Within the same field, [Kauer et al., 2014] proposed a control compensator design scheme for dropped packets under the weakly hard model that also guaranteed stability. Similarly, [Linsenmayer et al., 2021] used design techniques from optimal control theory to co-design controllers. [Caccamo et al., 2000] proposed to change the control task period according to the task execution time to improve schedulability and control performance. Finally, several works discussed the trade-offs between schedulability and control performance [Crespo et al., 1999; Eker and Cervin, 1999; Marti et al., 2001; Caccamo et al., 2002].

This paper distinguishes itself from previous synthesis literature as it assumes that a linear controller has already been developed. We propose an adaptive implementation of the control algorithm that does not require modifications to the system's design phase.

3.2 Research Problem Motivation

Despite the considerable amount of prior research, we argue that existing design approaches to handling real-time faults in controllers suffer from at least one of the following limitations:

(i) Assumptions about static controller – i.e., the controller cannot have any internal dynamics [Ramanathan, 1997; Schinkel and Chen, 2006; Zhang and Yu, 2010; Kumar et al., 2012; Kauer et al., 2014; Linsenmayer and Allgower, 2017; Pazzaglia et al., 2019; Maggio et al., 2020].

(ii) Complex co-design methodology, leading to conservative results – i.e., a detailed system model and a large extra design effort are needed [Marti et al., 2001; Schinkel and Chen, 2006; Kumar et al., 2012; Kauer et al., 2014; Linsenmayer and Allgower, 2017; Pazzaglia et al., 2021].

(iii) Increased runtime overhead – i.e., the approach reduces the likelihood of completing the control execution within the same total time budget [Crespo et al., 1999; Camacho et al., 2010; Caccamo et al., 2000].

The first limitation substantially narrows the applicability of the proposed design techniques, since the vast majority of real-world controllers include a dynamical part (e.g., an integrator to remove stationary errors, or a filter to estimate states or remove measurement noise). It is important to study dynamic controllers as they are more severely affected by computational faults than static ones. Intuitively, a static controller computes the control action u solely based on the latest measurement, and is therefore always up to date with respect to the plant state. In contrast, a dynamic controller computes u also with respect to its internal state, which is computed on the base of older measurements. When deadline misses occur, the controller state diverges from its desired value and degrades the control performance. We support this intuition with a motivating example, considering a static controller and a comparable dynamic controller that includes a Kalman filter [Åström and Wittenmark, 1984]. The two controllers are designed to give similar performance under ideal conditions. The example shows that, while the static controller inherently provides robustness to deadline misses, the dynamic controller is significantly more sensitive.

EXAMPLE 1—MOTIVATING EXAMPLE
Consider the discrete-time LTI plant

$$\mathcal{P}: \begin{cases} x_{k+1} = \begin{bmatrix} 0 & 1 \\ -0.8 & 1.8 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k \\ y_k = x_k. \end{cases} \tag{3}$$

The control system's objective is to bring the state x to zero. The plant $\mathcal{P}$ consists of a stable pole and an integrator, which is compatible with several real-world applications, e.g., a cart on a rail or a joint of a robotic arm. To regulate the plant, we first consider the static controller $\mathcal{C}_1$:

$$\mathcal{C}_1 : u_{k+1} = \begin{bmatrix} 0.256 & -0.372 \end{bmatrix} y_k.$$

The system is sampled and actuated using a sample time of $T = 0.1$ s. We consider a scenario where the ECU executing the control law is experiencing sporadic CPU overloads, resulting in 25% of the control jobs missing their corresponding deadlines. The initial state of the plant is $x_0 = [1,1]^{\mathrm{T}}$, and we want the controller to move it to the final position $x_f = [0,0]^{\mathrm{T}}$. In Figure 2, we show only the second component

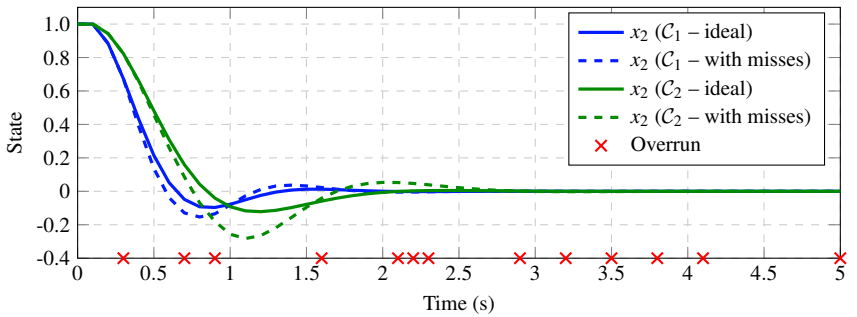


Figure 2. The state x_2 corresponding to $\mathcal{P}$ (sampled with $T = 0.1$s) being controlled by the static controller $\mathcal{C}_1$ (dashed blue) or the LQG controller $\mathcal{C}_2$ (dashed green) during an interval of sporadic overloads ($\times$). The corresponding ideal behaviours (system not subject to overruns) for the static (solid blue) and LQG (solid green) controllers are also plotted. The state x_1 follows a very similar trajectory and is not plotted for readability.

of the state vector x: the other component follows a very similar trajectory and is thus not plotted (for clarity). The behaviour of the plant controlled by $\mathcal{C}_1$ without deadline overruns is the solid blue line. The dashed blue line instead corresponds to the state evolution in the presence of deadline misses. The control task overruns are marked by red crosses on the horizontal axis. Despite the missed job completions, the plant state recovers gracefully in very few steps, similarly to the ideal behaviour of the controller (in the absence of overruns).

Practically, in all industrial applications, the control law $\mathcal{C}_1$ would be preceded by a noise filter or a state estimator, which introduces dynamic behaviour in the controller. Thus, we now consider the LQG controller $\mathcal{C}_2$, which contains a Kalman filter designed to suppress noise:

$$\mathcal{C}_2 : \begin{cases} z_{k+1} = \begin{bmatrix} -0.151 & 0.810 \\ -0.711 & 1.206 \end{bmatrix} z_k + \begin{bmatrix} 0.151 & 0.190 \\ 0.166 & 0.221 \end{bmatrix} y_k \\ u_{k+1} = \begin{bmatrix} 0.226 & -0.242 \end{bmatrix} z_k + \begin{bmatrix} -0.023 & -0.034 \end{bmatrix} y_k. \end{cases}$$

The dashed green line in Figure 2 shows the second component of the plant state x, controlled by the dynamic LQG controller $\mathcal{C}_2$ and subject to the same *deadline misses as controller $\mathcal{C}_1$. Comparing it to the ideal behaviour (solid green line), we observe a significant performance degradation compared to the static controller $\mathcal{C}_1$. This showcases that dynamic controllers suffer more under sporadic overloads than static controllers.*

As for the second limitation listed above, introducing additional complexity for the gain of better performance is not always a desired design solution. The more convoluted the control design, the higher the design cost. This is a consequence

of the increased development time needed to design the controller. When the complexity of the controller increases, the cost of the system upkeep increases due to the expert knowledge required. The most popular controllers in industry are thus simple ones that still perform adequately (e.g., the PI controller [Sun et al., 2016; Desborough and Miller, 2002; Åström and Hägglund, 1984]).

Considering the third limitation, introducing additional overhead for fault-prone systems risks even more deadline misses. Depending on the plant or task model, there are cases where increasing the overhead might be acceptable, e.g., if deadline misses appear sporadically in bursts. However, for many task models, increasing the execution time after a deadline miss could have severe consequences, causing domino effects where subsequent jobs also miss their deadlines.

One major strength of many previously proposed fault-tolerant control design methods is that they can guarantee closed-loop stability under their respective fault and system models [Schinkel and Chen, 2006; Kumar et al., 2012; Linsenmayer and Allgower, 2017; Linsenmayer et al., 2020]. A majority of the studies investigate static controllers. However, in most real-world applications, the controllers are dynamic, e.g., an integrator, estimator, or filter is included in the loop. The a priori guarantees provided for the static controller are hence lost. Additionally, if the model of the controlled plant is unknown, no a priori stability guarantees could be achieved. However, the resulting closed-loop system could be analysed a posteriori using any existing general method (see Section 2.3).

Research Objective Tackling all the shortcomings mentioned above in a holistic manner is a complex task that does not necessarily have a general solution. In this paper, we set out to derive a control adaptation strategy for *dynamic* controllers to handle deadline misses whilst providing a *simple* structure, *minimal* overhead, and that does not reduce the system performance under ideal conditions. To analyse the performance and robustness of the adaptive control strategy, we also seek to derive a stochastic mean-square analysis of the resulting closed-loop system to be performed a posteriori.

4. Real-Time Controller Adaptation

In this section, we derive an adaptive implementation scheme for real-time controllers to address the problems discussed in Section 3.2. We explain the intuition behind the proposed adaptation; then, starting from an arbitrary linear control algorithm, we show how to derive the adapted controller. The adaptive controller is complemented with a probabilistic analysis method to evaluate its effectiveness for a given control system and a given deadline miss model.

4.1 Adaptive Controller Synthesis

To simplify and generalise the adaptation approach, we do not assume any information about the control design, apart from the controller itself. In practice, this

implies that we consider neither the specific control design technique, the control system requirements, nor the plant's dynamics. The controller is assumed to be given in state-space form, specified by (2). This assumption is made without loss of generality, since any realisable linear digital controller can be expressed in this form [Åstrom and Murray, 2008].

The controller behaves *ideally* when every control period includes one job completing the execution of Equation (2), i.e., every job hits its deadline. By iterating the equation, the desired controller state and control action at any future time step can be computed. We formally define this behaviour as follows:

DEFINITION 3—IDEAL CONTROLLER
We denote the ideal controller*, $\mathcal{C}(q)$, as the discrete-time LTI controller $\mathcal{C}$ evolved over an interval of $q+1$ consecutive deadline hits. The controller state and output at the end of the interval are*

$$\mathcal{C}(q): \begin{cases} z_{k+q+1} = F^{q+1} z_k + \sum_{i=0}^{q} F^i G y_{k+q-i} \\ u_{k+q+1} = H F^q z_k + \\ \quad + H \sum_{i=1}^{q} \left[F^{i-1} G y_{k+q-i} \right] + K y_{k+q}. \end{cases} \tag{4}$$

With $q = 0$ we obtain the ideal controller behaviour over a single time step.

From the definition above, we see how the quantities of interest, z and u, are computed using the values of y in the interval $[k, k+q]$. These values correspond to the periodic measurements coming from the sensors. In the presence of deadline misses, the control task discards the measurements and does not update the controller's state. Thus, both the state and output of the controller deviate from their desired behaviours. Applying Equation (2) to a scenario of q consecutive deadline misses, we obtain the following:

DEFINITION 4—NOMINAL CONTROLLER
We denote the nominal controller*, $\mathcal{C}^n(q)$, as the discrete-time LTI controller $\mathcal{C}$ evolved over an interval of q consecutive deadline misses followed by one deadline hit. After the hit, the state and output are given by*

$$\mathcal{C}^n(q): \begin{cases} z_{k+q+1} = F z_k + G y_{k+q} \\ u_{k+q+1} = H z_k + K y_{k+q}. \end{cases} \tag{5}$$

In the absence of deadline misses, we have $\mathcal{C}^n(0) = \mathcal{C}$.

By comparing (4) and (5) we see that, when deadline misses occur, the controller state z_{k+q+1} and control action u_{k+q+1} diverge from the ideal values, i.e., the ones that would be obtained in the presence of only deadline hits. To compensate for this error (and consequently minimise it), we propose to dynamically alter the controller's computations in real time according to the number of deadline misses.

Ideally, we would like an adaptive controller $\mathcal{C}^a(q)$ that mimics the ideal controller $\mathcal{C}(q)$ for any q. However, this is infeasible due to how the controller's dynamics evolves during an interval of deadline misses. More specifically, Equation (4) shows that $\mathcal{C}(q)$ depends on the values of y_{k+i} for $i \in \{0,1,\ldots,q-1\}$. When the control task is subjected to faults, the corresponding jobs j_{k+i} miss their respective deadline. Thus, the unfinished jobs are terminated prematurely, a new job is released, and the corresponding measurement values y_{k+i} are lost. The nominal controller, after a series of misses, has access only to the sample y_{k+q}. This fundamentally limits how well the controller's ideal behaviour can be reconstructed.

Hence, we propose the use of an interpolation scheme to minimise the effect of the lost measurement values. We approximate the missing measurement values using a linear interpolation between y_{k-1} and y_{k+q} according to

$$\hat{y}_{k+q-i} = \frac{i\,y_{k-1} + (q+1-i)\,y_{k+q}}{q+1}. \tag{6}$$

If we substitute the values of y that are missing in the ideal controller, Equation (4), with the corresponding interpolated ones $\hat{y}$ from Equation (6), we obtain an adaptive controller as follows:

DEFINITION 5—ADAPTIVE CONTROLLER
We denote the adaptive controller, *$\mathcal{C}^a(q)$, as an adaptation of the controller $\mathcal{C}$ evolved over an interval of q consecutive deadline misses followed by one deadline hit. After the hit, the state and output are*

$$\mathcal{C}^a(q): \begin{cases} z_{k+q+1} = F_z(q)\,z_k + F_y(q)\,y_{k-1} + G_y(q)\,y_{k+q} \\ u_{k+q+1} = H_z(q)\,z_k + H_y(q)\,y_{k-1} + K_y(q)\,y_{k+q}, \end{cases} \tag{7}$$

where

$$\begin{aligned} F_z(q) &= F^{q+1} \\ F_y(q) &= \textstyle\sum_{i=0}^{q} \frac{i}{q+1} F^i G \\ G_y(q) &= \textstyle\sum_{i=0}^{q} \frac{q+1-i}{q+1} F^i G \\ H_z(q) &= H F^q \\ H_y(q) &= H \textstyle\sum_{i=1}^{q} \frac{i}{q+1} F^{i-1} G \\ K_y(q) &= K + H \textstyle\sum_{i=1}^{q} \frac{q+1-i}{q+1} F^{i-1} G. \end{aligned}$$

Note that for $q = 0$ we recover the original controller.

All matrices in $\mathcal{C}^a(q)$ ($q \in \{0,\ldots,q_{max}\}$) can be precomputed and stored in memory. Compared to the nominal controller, the two matrices F_y and H_y of size $n_z \times n_y$ and $n_u \times n_y$ need to be added to the controller, and one full set of controller matrices needs to be stored for each value of q. The memory requirement thus grows

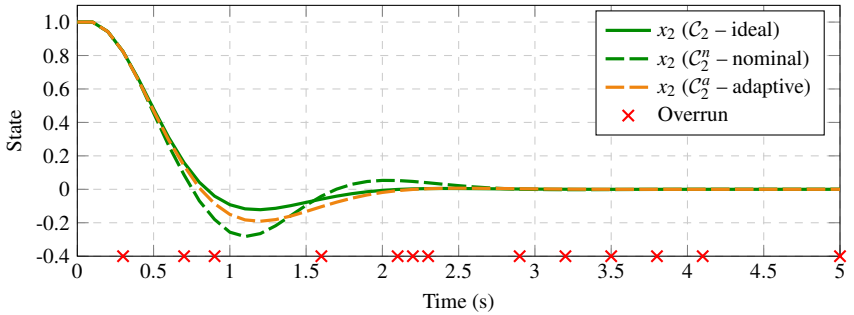


Figure 3. The state x_2 corresponding to $\mathcal{P}$ being controlled by the LQG controller $\mathcal{C}_2$ (dashed green) or adaptive LQG (dashed orange) controller during an interval of sporadic overloads ($\times$). The corresponding ideal behaviour (system not subject to overruns, solid green) is also plotted.

linearly with q_{max}. The adaptive controller also needs to store the old measurement value y_{k-1}, which makes it marginally more complex.

We now briefly discuss how the limitations described in Section 3.2 are addressed. This work proposes an adaptation of the F, G, H, and K matrices of the controller, thus adapting also the *dynamic* part of the controller. The adaptive controller's matrices can be precomputed directly from the specified controller using Equation (7), and then stored in memory for each $q \leq q_{max}$. The control algorithm can be developed independently of this adaptation, hence no extra design effort is needed. For $q = 0$ the adaptive and the ideal controllers are equivalent, i.e., $\mathcal{C}^a(0) = \mathcal{C}(0)$, thus guaranteeing that the controller's performance is not degraded under ideal conditions. Concerning the limitation of increased runtime overhead, each time a job is released, the counter q of immediately preceding deadline misses is evaluated and the corresponding controller matrices are selected. Hence, the execution time of the adaptive controller will be independent of q and only marginally longer than for the nominal controller.

In Section 3.2, we presented a motivating example of why it is insufficient to analyse static controllers subject to overruns. We conclude this section with a continuation of this example showing the benefits of our scheme.

EXAMPLE 2—APPLICATION TO MOTIVATING EXAMPLE
Consider the plant $\mathcal{P}$, from Equation (3)*, controlled by the LQG controller $\mathcal{C}_2$. We apply our proposed scheme to $\mathcal{C}_2$, obtaining the corresponding adaptive controller $\mathcal{C}_2^a(q)$. In Figure 3, we compare the plant state obtained by controlling the plant using the two controllers, subject to the same deadline misses as in Figure 2. The dashed green line and the dashed orange line correspond, respectively, to the nominal LQG (the same as in Figure 2) and adaptive LQG. The solid green line corresponds to the ideal controller's behaviour, i.e., in the absence of deadline misses. We*

note a significant improvement in the performance of the adaptive LQG compared to the nominal LQG: the oscillations disappear and the state converges faster.

The proposed adaptive controller is inspired by the ideal controller $\mathcal{C}(q)$, but there is no a priori guarantee that it will perform better than the nominal controller $\mathcal{C}^n(q)$. However, for a given plant and deadline miss model, the closed-loop system can be analysed and compared for different controllers. Since the actual sequence of hits and misses is generally unpredictable, in the next section, we propose a probabilistic method to compare the performance of the adaptive controller with the nominal controller.

4.2 Stochastic Performance Analysis

To analyse the performance of the closed-loop system, we need to model the plant, the external disturbance w, and the sequences of deadline misses. For what concerns the plant, we consider the standard state-space model presented in Equation (1). The plant disturbance w and the sequence of deadline hits and misses are modelled as stationary random processes with known statistical properties. This allows us to analyse the time-averaged performance of the closed-loop system subject to deadline misses. The analysis offered here is inspired by [Nilsson et al., 1998], which in turn relies on classical results for jump linear systems [Blair Jr. and Sworder, 1975].

The control performance is measured in terms of the weighted stationary variance of the plant output y,

$$J = \mathbb{E}\left[y_k^{\mathrm{T}} Q y_k\right], \tag{8}$$

where $Q \in \mathbb{R}^{n_y \times n_y}$ is a positive semidefinite weighting matrix[2]. A smaller value of the cost J is better, as it means that the controller can suppress the disturbance w more effectively. Additionally, the closed loop is guaranteed to be *stable* in the mean-square sense as long as J is finite [Fang and Loparo, 2002], i.e., $J < \infty$.

The state of the complete system consists of the plant and the controller in feedback interconnection, together with the buffered previous measurement value and the calculated next control signal. Accordingly, we define the closed-loop state vector as

$$\tilde{x}_k = \begin{bmatrix} x_k \\ z_k \\ y_{k-1} \\ u_k \end{bmatrix}.$$

As seen in Figure 1, the only external input of the closed-loop system is the disturbance w. Hence, the state of the closed loop evolves according to the time-varying linear system

$$\tilde{x}_{k+1} = \Phi_k \tilde{x}_k + \Gamma w_k. \tag{9}$$

[2] Without loss of generality, we assume that zero is the desired plant output. A non-zero reference value can be modelled as an exogenous signal that is stored in the plant and offset in the measurement signal.

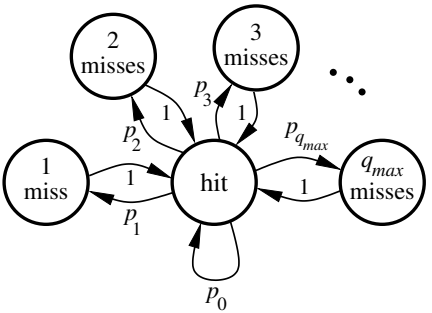


Figure 4. Markov model for the random sequence of hits and misses.

Here,

$$\Gamma = \begin{bmatrix} W \\ 0_{(n_z+n_y+n_u)\times n_w} \end{bmatrix},$$

while Φ_k can be either $\Phi_{hit}(q)$ or Φ_{miss}, depending on whether the current deadline was hit or missed, and defined as

$$\Phi_{hit}(q) = \begin{bmatrix} A & 0 & 0 & B \\ G_y(q)C & F_z(q) & F_y(q) & G_y(q)D \\ C & 0 & 0 & D \\ K_y(q)C & H_z(q) & H_y(q) & K_y(q)D \end{bmatrix},$$

where q denotes the number of consecutive deadline misses since the last hit, and

$$\Phi_{miss} = \begin{bmatrix} A & 0 & 0 & B \\ 0 & I_{n_z} & 0 & 0 \\ 0 & 0 & I_{n_y} & 0 \\ 0 & 0 & 0 & I_{n_u} \end{bmatrix}.$$

Assuming that w is a zero-mean white noise process with known covariance matrix $R = \mathbb{E}\left[w_k \cdot w_k^{\mathrm{T}}\right] \in \mathbb{R}^{n_w \times n_w}$, we can calculate how the state covariance, $P_k = \mathbb{E}\left[\tilde{x}_k \cdot \tilde{x}_k^{\mathrm{T}}\right]$, evolves over time. Evaluating the covariance of both sides in Equation (9) we obtain (see, e.g., [Åström and Wittenmark, 1984])

$$P_{k+1} = \Phi_k P_k \Phi_k^{\mathrm{T}} + \Gamma R \Gamma^{\mathrm{T}}. \tag{10}$$

We model the task execution as a random process, assuming that the pattern of hits and misses in the real-time system is described by the homogeneous Markov model shown in Figure 4. In this model, after each hit, the system will experience a miss interval of length $q \in \{0, \ldots, q_{max}\}$ with independent probability p_q. Naturally, $\sum_{i=0}^{q_{max}} p_i = 1$.

In each interval, the system will experience q deadline misses followed by one deadline hit. Iterating (10) over said interval, the covariance will then develop as

$$\begin{aligned} P_{k+q+1} = \Phi_{hit}(q)\Big(&(\Phi_{miss})^q P_k (\Phi_{miss}^{\mathrm{T}})^q \\ &+ \textstyle\sum_{i=0}^{q-1} (\Phi_{miss})^i \Gamma R \Gamma^{\mathrm{T}} (\Phi_{miss}^{\mathrm{T}})^i \Big) \Phi_{hit}^{\mathrm{T}}(q) \\ &+ \Gamma R \Gamma^{\mathrm{T}}. \end{aligned}$$

The time-varying closed-loop system together with the Markov model define a *discrete-time Markov jump linear system* for which well-established results exist (e.g., [Blair Jr. and Sworder, 1975; Nilsson et al., 1998; Lincoln and Cervin, 2002]). Using this theory, it is possible to calculate the *stationary* (time-averaged) state covariance, denoted $\bar{P}$. With this, the performance (8) can finally be obtained as

$$J = \operatorname{tr}\left(\bar{P}\,\bar{Q}\right),$$

where

$$\bar{Q} = \begin{bmatrix} C^{\mathrm{T}} Q C & 0 \\ 0 & 0_{n_z+n_y+n_u} \end{bmatrix}.$$

To compare the performance of different implementations, we first define the *ideal performance* as the cost J obtained when there are no deadline misses (i.e., $p_0 = 1$ and $p_i = 0, i \geq 1$). We then obtain the *relative performance degradation* of an arbitrary controller $\mathcal{C}^\dagger$ by calculating the weighted mean-square difference between the actual and ideal systems' outputs, $y_k^\dagger$ and y_k respectively, and normalising it with respect to the ideal performance J:

$$\frac{\Delta J^\dagger}{J} = \frac{\mathbb{E}\left[(y_k^\dagger - y_k)^{\mathrm{T}} Q (y_k^\dagger - y_k)\right]}{\mathbb{E}\left[y_k^{\mathrm{T}} Q y_k\right]}. \tag{11}$$

This can be found by analysing both systems in parallel when driven by the same noise sequence w_k:

$$\begin{bmatrix} \tilde{x}_{k+1}^\dagger \\ \tilde{x}_{k+1} \end{bmatrix} = \begin{bmatrix} \Phi_k^\dagger & 0 \\ 0 & \Phi_k \end{bmatrix} \begin{bmatrix} \tilde{x}_k^\dagger \\ \tilde{x}_k \end{bmatrix} + \begin{bmatrix} \Gamma \\ \Gamma \end{bmatrix} w_k$$

After finding the stationary state covariance $\bar{P}_e$ of this extended system (using the same technique as referred to above), we can retrieve the absolute performance difference as

$$\Delta J^\dagger = \operatorname{tr}\left(\bar{P}_e \begin{bmatrix} \bar{Q} & -\bar{Q} \\ -\bar{Q} & \bar{Q} \end{bmatrix}\right).$$

5. Experimental Evaluation

In this section, we compare our adaptive controller with the nominal controller implementation for different case studies. We demonstrate the practical usefulness of the proposed controller by examining its impact on real hardware, namely, a ball and beam plant. We compare the performance of the adaptive control system with the nominal one, according to the analysis presented in Section 4.2. Finally, we complement the results with a worst-case switching stability analysis of the nominal and adaptive controlled systems.

In addition to the evaluation on the physical system, we present aggregate results obtained from a set of control benchmarks, representative of the process industry. We use this set of plants to evaluate the general applicability of our approach. To make the evaluation comprehensive, we chose an unstable plant (the ball and beam) for the physical experiments and a set of mainly stable plants for the aggregate results. Furthermore, we remark that all the considered controllers are dynamic. As discussed in Section 3.1, to the best of our knowledge, only one previous work considers dynamic controllers [Pazzaglia et al., 2021]. In that work, however, a different overrun handling method is used, and a proper comparison is therefore not possible.

5.1 Real World Evaluation – Ball and Beam

System Description and Models The ball and beam [Wellstead et al., 1978] is a common example in the automatic control literature and education, where a ball is free to roll over a beam that in turn is tilted by a servo motor. The control objective is to make the ball position follow a reference trajectory across the beam by adjusting the voltage sent to the motor. Both the beam angle and the ball position can be measured. Assuming the sampling period $T = 0.01$ s, a discrete-time plant model $\mathcal{P}$ was derived as

$$\mathcal{P}: \begin{cases} x_{k+1} = \begin{bmatrix} 1 & 0.015 & 0.0003 & 0 \\ 0 & 1 & 0.045 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 2.9 \cdot 10^{-5} \\ 0.0058 \\ 0.256 \\ 0 \end{bmatrix} u_k + w_k \\ y_k = \begin{bmatrix} 0.5 & 0 & 0 & -1 \\ 0 & 0 & 0.25 & 0 \end{bmatrix} x_k, \end{cases}$$

where the four components of x_k represent the ball position, ball velocity, beam velocity, and ball reference, respectively. The external signal vector w_k is assumed to be white noise with variance $R = \operatorname{diag}\{1,1,1,1\}$. Under this state-space model, the objective is to regulate both outputs y_k to zero, with the performance weighting matrix $Q = \operatorname{diag}\{1,1\}$.

To control $\mathcal{P}$ we design a cascaded P–PID controller. Cascaded controllers are frequently applied to systems with multiple measurements where one measured quantity affects another, but not vice versa. Thus, the plant measurements

Table 1. Analytical study of the relative performance degradation of the ball and beam plant $\mathcal{P}$ using either the nominal $\mathcal{C}^n$ or adaptive controller $\mathcal{C}^a$.

p	10%	20%	30%	40%	50%	60%	70%
$\Delta J^n/J$	2.5%	9.2%	20.8%	39.9%	75.3%	156%	452%
$\Delta J^a/J$	0.1%	0.1%	0.3%	0.6%	1.1%	2.5%	6.7%

can be controlled in sequential order (hence the naming *cascaded*) using a controller designed for each measurement signal. In our case study, this is implemented by a proportional (P) controller designed for controlling the beam's angle and a proportional–integral–derivative (PID) controller for the ball's position. The controller is run as a periodically executing task with period $T = 0.01$ s on a single core CPU where overrun deadlines are killed and the corresponding sensor data is discarded. In between actuator calls, the control signal is assumed to be held constant. The state-space representation of our controller is

$$\mathcal{C}: \begin{cases} z_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 0.9685 \end{bmatrix} z_k + \begin{bmatrix} 0.025 & 0 \\ -0.2608 & 0 \end{bmatrix} y_k, \\ \\ u_{k+1} = \begin{bmatrix} -0.108 & -0.2608 \end{bmatrix} z_k + \begin{bmatrix} -2.43 & -3 \end{bmatrix} y_k. \end{cases}$$

Experiments Design We apply the performance analysis presented in Section 4.2 to the plant model $\mathcal{P}$ controlled using either the ideal ($\mathcal{C}$), nominal ($\mathcal{C}^n$), or adaptive ($\mathcal{C}^a$) implementations from Section 4.1. We include the effect of deadline misses only on the nominal and adaptive control systems. The probability distribution p_q can be chosen arbitrarily according to the desired task model. For simplicity, we assume here that the deadline misses are Bernoulli distributed [Schenato et al., 2007], i.e., the probabilities of missing deadlines in each period are independently and identically distributed with probability p. This results in the probability $p_q = (1-p)p^q$ of q consecutive deadline misses followed by a hit. We assume that no more than $q_{max} = 20$ consecutive deadlines can be missed. The latter assumption might seem restrictive, but if the probability of missing a deadline is 30%, the probability of missing 20 consecutive deadlines is less than $4 \cdot 10^{-11}$.

We measure the relative performance of the nominal and adaptive controllers according to the quantity $\Delta J^{\dagger}/J$ in Equation (11). Since the mean-square deviation from the ideal controller is used to evaluate the relative performance, the *optimal* achievable cost is 0. For the real system, we do not feed the system with white noise, but we expose the system to a repeatable exogenous signal in the form of periodic reference changes. Furthermore, we evaluate the relative performance degradation empirically from the measured signals using Equation (11), with $\mathbb{E}$ being interpreted as the mean value of the real signals.

Table 2. Empirical study of the relative performance degradation of the real ball and beam plant using either the nominal $\mathcal{C}^n$ or adaptive controller $\mathcal{C}^a$.

p	10%	20%	30%	40%	50%	60%	70%
$\Delta J^n/J$	6.3%	27.1%	22.5%	50.5%	73.1%	260%	∞
$\Delta J^a/J$	6.1%	7.8%	3.2%	4.6%	3.8%	4.9%	11.7%

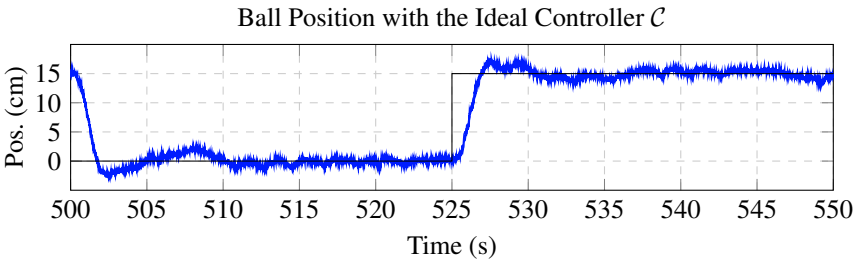


Figure 5. Snippet of the test performed on the real ball and beam plant using the ideal controller, i.e., without deadline misses. The plot shows one period of the square wave used as reference, the black line. The blue line shows the ball's position.

To complement the performance analysis, we perform a JSR stability analysis on the model to determine the maximum number of consecutive deadline misses that are tolerated while still guaranteeing closed-loop stability.

Analytical Evaluation The performance results obtained with the analytical study of $\mathcal{P}$ for different values of p are summarised in Table 1. From the table, we can see that the adaptive controller drastically improves the relative performance (in comparison to the nominal controller) across all deadline miss probabilities. Already for small probabilities, the nominal controller significantly degrades the relative performance compared to the ideal controller; e.g., for $p = 30\%$ the relative performance is degraded by 20.8%. This can be compared to the adaptive controller, where the relative performance reduction stays below 5% until the miss probability reaches 70%.

Analysing the switching stability, we calculated the JSR for the set of closed-loop matrices corresponding to $i = \{0, 1, \ldots, q\}$ consecutive deadline misses followed by one hit ($q \leq q_{max}$). The nominal control system is guaranteed to be switching stable (i.e., the JSR is below 1) for a maximum of $q = 2$ consecutive deadline misses, while the adaptive control system is guaranteed stable up to $q = 8$. We conclude that the adaptive controller improves also worst-case robustness against deadline misses for the ball and beam. However, we emphasise that these results do *not* imply that the system will go unstable if more deadline misses occurs; only that

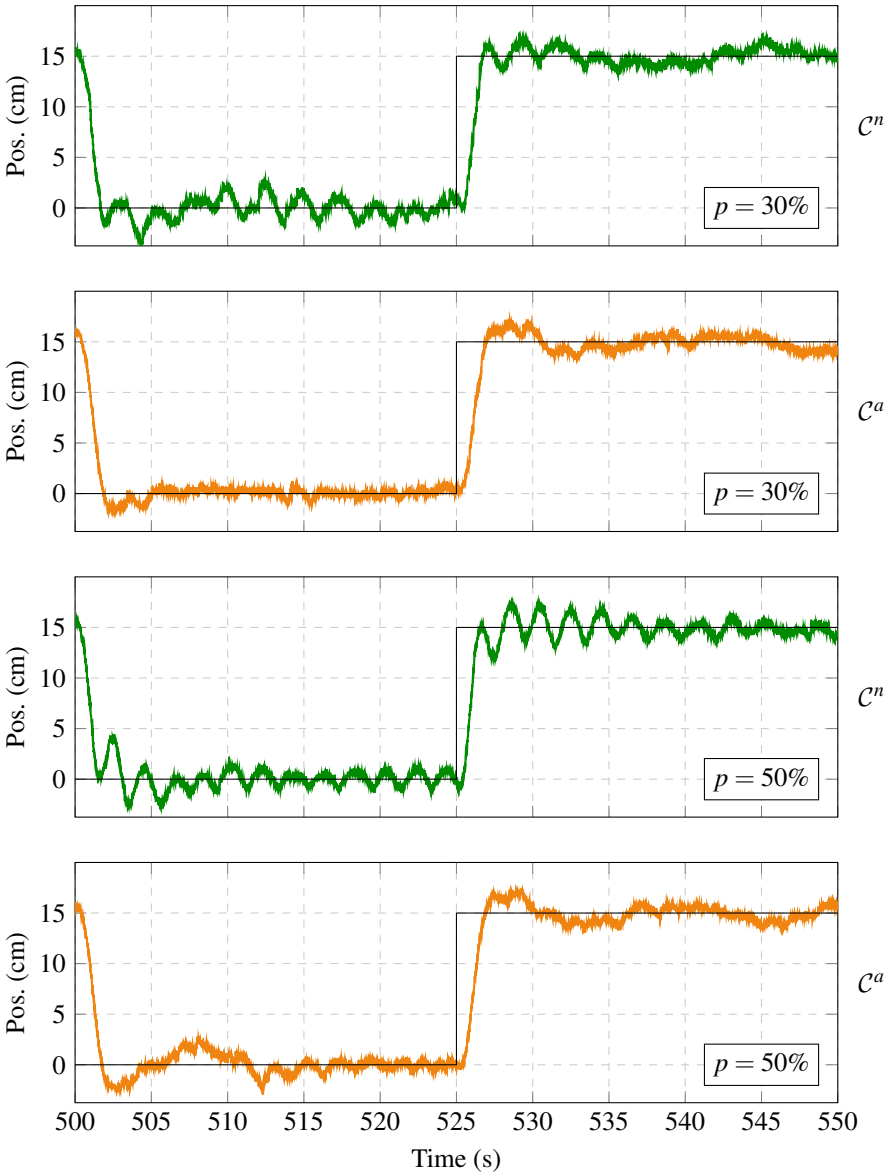


Figure 6. Snippets of the tests performed on the real ball and beam plant for $p = 30\%$ (two top plots) and $p = 50\%$ (two bottom plots). The plots show one period of the square wave used as reference, the black line. The coloured lines show the ball position, in green for the nominal controller (first and third plots) and orange for the adaptive controller (second and third plots).

the system is *guaranteed* switching stable if no more than q consecutive deadline misses are ever experienced.

Empirical Evaluation We conducted experiments on the physical ball and beam plant to evaluate the performance of the controller on a real system.[3] Each experiment is run for 10 minutes, where the control objective is for the ball to follow a square-wave reference across the beam. The square wave has a period of 50 s and alternates between position 0 and 15 cm. Differently from the analytical evaluation, it is impossible to obtain the same exogenous signal w_k in the different experiments. While the reference changes can be exactly repeated, the real stochastic disturbances (in the form of electrical noise, mechanical glitches, etc.) are not repeatable. This means that the empirical cost relative to the ideal case, as measured by Equation (11), is not expected to be zero even in the complete absence of deadline misses.

Figure 5 displays a snippet of the ball's position (blue line) under said ideal conditions. The ball quite successfully follows the reference (black line). Here, the fluctuations around the reference are caused by measurement noise and irregularities in the beam surface, where the latter can cause the ball to get lodged in an undesired position and thus result in oscillations.

After measuring the performance of the ideal controller, each controller ($\mathcal{C}^n$ and $\mathcal{C}^a$) was applied to the system, using probabilities $p \in \{10\%, 20\%, \ldots, 70\%\}$ of missing each deadline (with $q_{max} = 20$). The results of the experiments are reported in Table 2, where the relative performance degradation $^{\Delta J^\dagger}/_{J}$ is computed for both the nominal and adaptive controllers. To give an intuition for how the physical system behaves, in Figure 6 we provide a snippet of a time plot portraying the ball's position controlled by either the nominal (upper plots) or adaptive (lower plots) controller. We distinguish the differences between the nominal and adaptive controllers for a probability $p = 30\%$ (left plots) of missing a deadline. The nominal controller shows oscillations around the reference value. When the probability of missing a deadline is increased to $p = 50\%$ (right plots), the nominal controller's oscillations grow more evident, while the adaptive controller appears unaffected (compared to the ideal controller in Figure 5).

From Table 2, we observe that the adaptive controller has a lower performance degradation across all deadline miss probabilities $p \geq 20\%$ compared to the nominal controller. The performance of the adaptive controller seems virtually unaffected for $p \leq 60\%$, where the baseline relative degradation of approximately 4% to 8% is due to the natural disturbances in the system. The nominal controller on the other hand experiences significant performance degradation at higher miss probabilities, and for $p = 70\%$ the system becomes unstable – we report this as an infinite cost.

[3] A video, showing experiments with the real ball and beam system can be viewed at `https://youtu.be/6y_C7NIzXto`. The video provides a real-world comparison between the nominal and adaptive controllers for $p = \{30\%, 50\%, 70\%\}$.

In summary, both the analytical and empirical studies show that the adaptive controller $\mathcal{C}^a$ consistently outperforms the nominal controller $\mathcal{C}^n$ for the ball and beam. Furthermore, the adaptive controller can tolerate a large likelihood of random deadline misses (at least 60%) without any noticeable performance degradation.

5.2 Benchmark Evaluation – Process Industry

System Description and Models To evaluate the general applicability of the proposed adaptive controller, we perform an extensive evaluation campaign on a benchmark set of plants. The set was developed specifically to evaluate various PID designs [Åström and Hägglund, 2004] in the process industry. It consists of 134 unique plants separated into 9 categories, where each category has its own specific properties frequently recognised in the process industry. Since the benchmark was developed specifically with process industrial plants in mind, the majority of the plants are stable, i.e., all their eigenvalues lie inside the unit circle. However, there are also plants with integrating dynamics included in the benchmark, i.e., an eigenvalue in 1; these plants are generally not considered stable. For each plant, two controllers – a PI and a PID controller – are optimised using known methods [Garpinger and Hägglund, 2008]; hence, 268 unique control systems are analysed in total.

Experiments Design Similarly to the ball and beam, we analyse the relative performance of the nominal and adaptive controllers in accordance with the analysis described in Section 4.2. We again consider the probability of missing a deadline to follow a Bernoulli distribution with probabilities $p \in \{10\%, 30\%, 50\%, 70\%\}$ and a maximum of $q_{max} = 20$ consecutive deadline misses. We feed the systems with a stochastic disturbance and analytically evaluate the ability of the controllers to reject it. Differently from the ball and beam, we analyse the systems when subject to brown noise, i.e., integrated white noise [Schmidt, 1985]. The brown noise model is generally considered appropriate for process industrial plants since it is dominant for low frequencies (e.g., load disturbances and disturbances from nearby heavy machinery). We assume that the *same* disturbance process enters the ideal, nominal, and adaptive control systems; this guarantees an unbiased comparison between the different controllers. For each of the 268 control systems we calculate the relative performance $\Delta J^\dagger / J$ for both the nominal and the adaptive controller.

Similarly to the ball and beam, we complement our performance analysis with a JSR worst-case stability analysis.

Experiments Results In Figure 7 we display histograms reporting the relative performance degradation of all the 268 control systems. The horizontal axis displays the relative performance $\Delta J^\dagger / J$ in logarithmic scale. The vertical axis counts the number of control systems with a given relative performance. The four plots correspond to the different deadline miss probabilities considered. In each plot, we represent the nominal controllers with green bars and the adaptive controllers with orange bars. Unstable closed-loop systems have an infinite cost and are thus marked in the rightmost part of the plot, beyond the red dashed threshold.

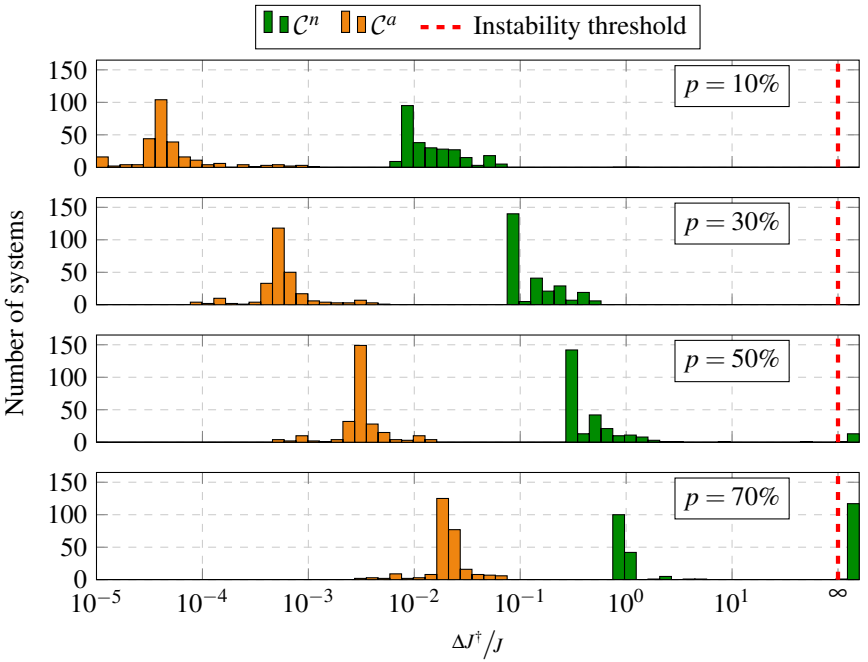


Figure 7. Histograms comparing the relative performance degradation of the nominal and adaptive controllers for the benchmark plants. The plots correspond to different deadline miss probabilities p. The orange bars report the performance obtained with the adaptive controller $\mathcal{C}^a$, while the green bars report the performance obtained with the nominal controller $\mathcal{C}^n$. The systems with a performance worse than the stability threshold (red dashed line) resulted in unstable dynamics.

From Figure 7 we see that the adaptive controller performs better than the nominal one for *all* the 268 control systems, regardless of the probability of missing a deadline. Despite the control systems' dynamics varying significantly (e.g., lag dominated, lead dominated, oscillatory, high system order, integrating), the worst adaptive control system still performs better than the best nominal control system for all p. The improvement is particularly distinguishable for lower probabilities, e.g., $p = 10\%$, where the mean relative cost over all the control systems is improved by two orders of magnitude.

Second, when the probability of missing a deadline grows, the relative performance degradation increases accordingly. For $p = 50\%$ and $p = 70\%$ some of the systems using the nominal controller become unstable, i.e., $\Delta J^\dagger / J = \infty$. In the case of $p = 70\%$, more than 40% of the nominal control systems are unstable. On the other hand, *all* the adaptive control systems are stable and have a relative cost degradation below 10%. This suggests that $\mathcal{C}^a$ improves both performance and robustness

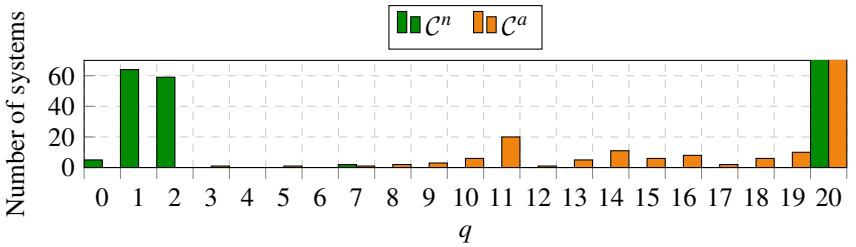


Figure 8. Histogram reporting the number of benchmark systems (out of 268) that are guaranteed switching stable for up to q consecutive deadline misses, according to the JSR analysis. For each value of q, the green bar (left) reports how many $\mathcal{C}^n$ controlled systems can tolerate up to q consecutive deadline misses and the orange bar (right) reports the corresponding number of $\mathcal{C}^a$ controlled systems. For readability the y axis is cut at 65: a total of 138 plants can tolerate 20 or more misses with the nominal controller, and a total of 185 plants can tolerate 20 or more misses with the adaptive controller.

compared to the nominal controller.

To verify that the adaptive controller improves the robustness to deadline misses compared to the nominal controller, we complement the evaluation with a JSR analysis. The histogram in Figure 8 shows, for each value of q, the number of control systems that are guaranteed switching stable for a maximum number of consecutive deadline misses q, when they are controlled with either the nominal ($\mathcal{C}^n$) or the adaptive ($\mathcal{C}^a$) controller. Intuitively, the more control systems that can guarantee switching stability for a higher value of q, the better. The vertical axis of the histogram is cut at 65 for legibility: this affects only the columns for $q = 20$ where the nominal controller can guarantee stability for 138 plants while the adaptive controller can guarantee stability for 185 plants.

For the nominal controller, we see that the maximum number of consecutive deadline misses tolerated by the system varies greatly between the different control systems. In the whole benchmark, 138 systems were stable for (at least) 20 consecutive deadline misses, but 123 systems were guaranteed stable only for one or two misses. Furthermore, 5 of the nominal control systems were unstable unless *all* of the control task's deadlines were hit.

For the adaptive controller, on average, a much larger number of consecutive deadline misses can be tolerated. Out of all the control systems, 185 were stable for (at least) $q = 20$, and the large majority of the remaining control systems are guaranteed to tolerate between $q = 10$ and $q = 19$ consecutive deadline misses. Additionally, we see that *all* adaptively controlled systems can tolerate at least 3 deadline misses.

We note that both for the nominal and adaptive controllers, a significant number of control systems are stable for 20 deadline misses. This presumably follows from

the (mainly) stable nature of the plants in the benchmark, an attribute that generally makes the system more robust.

The results of the evaluation campaign confirm the hypothesis that the proposed adaptive controller improves the control system's performance in the presence of deadline misses. While we observed some cases in which the nominal controller goes unstable and the adaptive controller is stable, we never observed the opposite. Additionally, the adaptive controller does not compromise the performance under ideal conditions, and it preserves the major part of the ideal controller's performance when deadline misses are present.

6. Conclusion

In this paper, we have proposed a novel adaptive implementation scheme for real-time embedded controllers, aiming to increase their robustness to arbitrary patterns of deadline misses. The approach adapts a general, predesigned linear controller according to the number of consecutive deadline overruns. No additional assumptions are made on the predesigned controller or the plant model, and only minimal requirements are put on the control task (Logical Execution Time) and the real-time operating system (late tasks being killed). The adaptation scheme can be implemented without any additional complexity or design overhead, hence strengthening the industrial applicability. Additionally, we extend the state-of-the-art by considering dynamic controllers.

To analyse our controller implementation, we developed a probabilistic approach. With said approach, it is possible to leverage a plant model and a probabilistic model of deadline misses to evaluate the effectiveness of the adaptive controller. We complement this average-case analysis with a worst-case stability analysis based on JSR. We used both approaches to evaluate the controller on both a physical plant and a considerable number of simulated plants from the literature on process control. The results show that our adaptive controller implementation consistently improves system performance and robustness to deadline misses.

Acknowledgements

This work was supported by the ELLIIT Strategic Research Area. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project). This (publication/report) reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

References

Akesson, B., M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis (2020). "An empirical survey-based study into industry practice in real-time systems". In: *41st IEEE Real-Time Systems Symposium*.

Aminifar, A., S. Samii, P. Eles, and Z. Peng (2011). "Control-quality driven task mapping for distributed embedded control systems". In: *17th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 133–142. DOI: 10.1109/RTCSA.2011.41.

Årzén, K.-E., A. Cervin, J. Eker, and L. Sha (2000). "An introduction to control and scheduling co-design". In: *39th IEEE Conference on Decision and Control*, pp. 4865–4870. DOI: 10.1109/CDC.2001.914701.

Åstrom, K. J. and R. M. Murray (2008). *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, USA. ISBN: 0691135762.

Åström, K. J. and T. Hägglund (1984). "Automatic tuning of simple regulators with specifications on phase and amplitude margins". *Automatica* **20**:5, pp. 645–651. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(84)90014-1.

Åström, K. J. and T. Hägglund (2004). "Revisiting the Ziegler-Nichols step response method for PID control". *Journal of Process Control* **14**:6, pp. 635–650. ISSN: 0959-1524.

Åström, K. J. and B. Wittenmark (1984). *Computer controlled systems: Theory and design (1 ed.)* English. Prentice-Hall. ISBN: 0-13-164319-3.

Baruah, S. K. and J. R. Haritsa (1997). "Scheduling for overload in real-time systems". *IEEE Transactions on Computers* **46**:9, pp. 1034–1039. DOI: 10.1109/12.620484.

Bernat, G., A. Burns, and A. Liamosi (2001). "Weakly hard real-time systems". *IEEE Transactions on Computers* **50**:4, pp. 308–321. DOI: 10.1109/12.919277.

Blair Jr., W. P. and D. D. Sworder (1975). "Feedback control of a class of linear discrete systems with jump parameters and quadratic cost criteria". *International Journal of Control* **21**:5, pp. 833–841. DOI: 10.1080/00207177508922037.

Caccamo, M., G. Buttazzo, and Lui Sha (2000). "Elastic feedback control". In: *12th Euromicro Conference on Real-Time Systems*, pp. 121–128. DOI: 10.1109/EMRTS.2000.853999.

Caccamo, M., G. Buttazzo, and L. Sha (2002). "Handling execution overruns in hard real-time control systems". *IEEE Transactions on Computers* **51**:7, pp. 835–849. DOI: 10.1109/TC.2002.1017703.

Camacho, E. F., T. Alamo, and D. M. de la Peña (2010). "Fault-tolerant model predictive control". In: *15th IEEE Conference on Emerging Technologies in Factory Automation*, pp. 1–8. DOI: 10.1109/ETFA.2010.5641226.

Cervin, A. (2005). "Analysis of overrun strategies in periodic control tasks". *IFAC Proceedings Volumes* **38**:1. 16th IFAC World Congress, pp. 219–224. ISSN: 1474-6670. DOI: 10.3182/20050703-6-CZ-1902.01076.

Cervin, A., B. Lincoln, J. Eker, K.-E. Årzén, and G. Buttazzo (2004). "The jitter margin and its application in the design of real-time control systems". In: *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*.

Chen, K.-H. and J.-J. Chen (2017). "Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors". In: *12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 1–8. DOI: 10.1109/SIES.2017.7993392.

Chen, K.-H., G. Von Der Brüggen, and J.-J. Chen (2018). "Analysis of deadline miss rates for uniprocessor fixed-priority scheduling". In: *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 168–178. DOI: 10.1109/RTCSA.2018.00028.

Crespo, A., I. Ripoll, and P. Albertos (1999). "Reducing delays in RT control: the control action interval". In: *14th IFAC World Congress 1999, Beijing, China*. DOI: https://doi.org/10.1016/S1474-6670(17)57454-6.

Desborough, L. and R. Miller (2002). "Increasing customer value of industrial control performance monitoring—Honeywell's experience". *AIChE Symposium Series* **98**.

Eker, J. and A. Cervin (1999). "A Matlab toolbox for real-time and control systems co-design". In: *6th International Conference on Real-Time Computing Systems and Applications*. DOI: 10.1109/RTCSA.1999.811266.

Ernst, R., S. Kuntz, S. Quinton, and M. Simons (2018). "The logical execution time paradigm: new perspectives for multicore systems". *Dagstuhl Reports* **8**, pp. 122–149.

Fang, Y. and K. Loparo (2002). "Stochastic stability of jump linear systems". *IEEE Transactions on Automatic Control* **47**:7, pp. 1204–1208. DOI: 10.1109/TAC.2002.800674.

Frehse, G., A. Hamann, S. Quinton, and M. Woehrle (2014). "Formal analysis of timing effects on closed-loop properties of control software". In: *35th IEEE Real-Time Systems Symposium (RTSS)*, pp. 53–62.

Garpinger, O. and T. Hägglund (2008). "A software tool for robust PID design". *IFAC Proceedings Volumes* **41**:2. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20080706-5-KR-1001.01082.

Gupta, R. A. and M. Chow (2010). "Networked control system: overview and research trends". *IEEE Transactions on Industrial Electronics* **57**:7, pp. 2527–2535. DOI: 10.1109/TIE.2009.2035462.

Hägglund, T. and K. J. Åström (1983). *A method and an apparatus in tuning a PID-regulator*. Patent application WO/1983/000753.

Hammadeh, Z. A. H., S. Quinton, R. Henia, L. Rioux, and R. Ernst (2017). "Bounding deadline misses in weakly-hard real-time systems with task dependencies". In: *Design Automation and Test in Europe (DATE)*. Lausanne, Switzerland. URL: http://ieeexplore.ieee.org/document/7927054/.

Henzinger, T., B. Horowitz, and C. Kirsch (2003). "Giotto: A time-triggered language for embedded programming". *Proceedings of the IEEE* **91**:1, pp. 84–99. DOI: 10.1109/JPROC.2002.805825.

Hertneck, M., S. Linsenmayer, and F. Allgöwer (2019). "Nonlinear dynamic periodic event-triggered control with robustness to packet loss based on non-monotonic lyapunov functions". In: *58th IEEE Conference on Decision and Control (CDC)*, pp. 1680–1685.

Hertneck, M., S. Linsenmayer, and F. Allgöwer (2020). "Stability analysis for nonlinear weakly hard real-time control systems". *IFAC-PapersOnLine* **53**:2. 21st IFAC World Congress, pp. 2594–2599. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.307.

Hertneck, M., S. Linsenmayer, and F. Allgöwer (2021). "Efficient stability analysis approaches for nonlinear weakly-hard real-time control systems". *Automatica* **133**, p. 109868. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2021.109868. URL: https://www.sciencedirect.com/science/article/pii/S0005109821003903.

Hespanha, J. P., P. Naghshtabrizi, and Y. Xu (2007). "A survey of recent results in networked control systems". *Proceedings of the IEEE* **95**:1, pp. 138–162. DOI: 10.1109/JPROC.2006.887288.

Kauer, M., D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy (2014). "Fault-tolerant control synthesis and verification of distributed embedded systems". In: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6. DOI: 10.7873/DATE.2014.069.

Kirsch, C. and A. Sokolova (2012). "The logical execution time paradigm". In: *Advances in Real-Time Systems*. Springer Berlin Heidelberg, pp. 103–120. ISBN: 978-3-642-24349-3.

Kumar, P., D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele (2012). "A hybrid approach to cyber-physical systems verification". In: *49th Annual Design Automation Conference (DAC)*. San Francisco, California, pp. 688–696. ISBN: 9781450311991. DOI: 10.1145/2228360.2228484. URL: https://doi.org/10.1145/2228360.2228484.

Liberzon, D. (2003). *Switching in Systems and Control*. Systems & control. Birkhauser. ISBN: 9783764342975.

Lincoln, B. and A. Cervin (2002). "Jitterbug: a tool for analysis of real-time control performance". In: *IEEE Conference on Decision and Control*. Vol. 2, pp. 1319–1324. DOI: 10.1109/CDC.2002.1184698.

Linsenmayer, S. and F. Allgower (2017). “Stabilization of networked control systems with weakly hard real-time dropout description”. In: *56th IEEE Conference on Decision and Control (CDC)*, pp. 4765–4770.

Linsenmayer, S., M. Hertneck, and F. Allgower (2020). “Linear weakly hard real-time control systems: time- and event-triggered stabilization”. *IEEE Transactions on Automatic Control*.

Linsenmayer, S., B. W. Carabelli, S. Wildhagen, K. Rothermel, and F. Allgöwer (2021). “Controller and triggering mechanism co-design for control over time-slotted networks”. *IEEE Transactions on Control of Network Systems* **8**, pp. 222–232.

Lozoya, C., P. Mart, M. Velasco, J. M. Fuertes, and E. X. Martin (2013). “Resource and performance trade-offs in real-time embedded control systems”. *Real-Time Systems* **49**:3, pp. 267–307. DOI: 10.1007/s11241-012-9174-9.

Maggio, M., A. Hamann, E. Mayer-John, and D. Ziegenbein (2020). “Control-system stability under consecutive deadline misses constraints”. In: *32nd Euromicro Conference on Real-Time Systems (ECRTS)*. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Marchand, A. and M. Chetto (2008). “Dynamic scheduling of periodic skippable tasks in an overloaded real-time system”. In: *2008 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pp. 456–464. DOI: 10.1109/AICCSA.2008.4493573.

Markovi, F., A. V. Papadopoulos, and T. Nolte (2021). “On the convolution efficiency for probabilistic analysis of real-time systems”. In: Brandenburg, B. B. (Ed.). *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 16:1–16:22. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.16. URL: https://drops.dagstuhl.de/opus/volltexte/2021/13947.

Marti, P., J. M. Fuertes, G. Fohler, and K. Ramamritham (2001). “Jitter compensation for real-time control systems”. In: *22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, pp. 39–48. DOI: 10.1109/REAL.2001.990594.

Milligan, M. and H. Cragon (1996). “The use of cache memory in real-time systems”. *Control Engineering Practice* **4**:10, pp. 1435–1442. ISSN: 0967-0661. DOI: https://doi.org/10.1016/0967-0661(96)00154-2.

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998). “Stochastic analysis and control of real-time systems with random time delays”. *Automatica* **34**:1, pp. 57–64. ISSN: 0005-1098. DOI: 10.1016/S0005-1098(97)00170-2.

Oshana, R. (2006). "Overview of embedded systems and real-time systems". In: Oshana, R. (Ed.). *DSP Software Development Techniques for Embedded and Real-Time Systems*. Embedded Technology. Newnes, Burlington, pp. 19–34. ISBN: 978-0-7506-7759-2. DOI: https://doi.org/10.1016/B978-075067759-2/50004-1. URL: https://www.sciencedirect.com/science/article/pii/B9780750677592500041.

Pazzaglia, P., A. Hamann, D. Ziegenbein, and M. Maggio (2021). "Adaptive design of real-time control systems subject to sporadic overruns". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*.

Pazzaglia, P., C. Mandrioli, M. Maggio, and A. Cervin (2019). "DMAC: Deadline-Miss-Aware Control". In: *31st Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 1:1–1:24. ISBN: 978-3-95977-110-8.

Ramamritham, K. (1996). "Where do time constraints come from? Where do they go?" *International Journal of Database Management* **7**:2, pp. 4–11.

Ramanathan, P. (1997). "Graceful degradation in real-time control applications using (m,k)-firm guarantee". In: *27th IEEE International Symposium on Fault Tolerant Computing*, pp. 132–141.

Rehbinder, H. and M. Sanfridson (2000). "Integration of off-line scheduling and optimal control". In: *12th Euromicro Conference on Real-Time Systems*, pp. 137–143.

Rota, G.-C. and W. G. Strang (1960). "A note on the joint spectral radius". In: *Proceedings of the Netherlands Academy*.

Schenato, L. (2009). "To zero or to hold control inputs with lossy links?" *IEEE Transactions on Automatic Control* **54**:5, pp. 1093–1099.

Schenato, L., B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry (2007). "Foundations of control and estimation over lossy networks". *Proceedings of the IEEE* **95**:1, pp. 163–187. DOI: 10.1109/JPROC.2006.887306.

Schinkel, M. and W.-H. Chen (2006). "Control of sampled data systems with variable sampling rate". *International Journal of Systems Science* **37**. DOI: 10.1080/00207720600681161.

Schmidt, P. C. (1985). "Handbook of stochastic methods for physics, chemistry and the natural sciences". *Berichte der Bunsengesellschaft für physikalische Chemie* **89**:6, pp. 721–721.

Seong Woo Kwak, Byung Jae Choi, and Byung Kook Kim (2001). "An optimal checkpointing-strategy for real-time control systems under transient faults". *IEEE Transactions on Reliability* **50**:3, pp. 293–301. DOI: 10.1109/24.974127.

Stankovic, J., M. Spuri, M. D. Natale, and G. Buttazzo (1995). "Implications of classical scheduling results for real-time systems". *Computer* **28**:6, pp. 16–25. DOI: 10.1109/2.386982.

Steinbauer, G. (2013). "A survey about faults of robots used in RoboCup". In: Chen, X. et al. (Eds.). *RoboCup 2012: Robot Soccer World Cup XVI*. Springer, Berlin, Heidelberg, pp. 344–355. ISBN: 978-3-642-39250-4.

Sun, L., D. Li, and K. Y. Lee (2016). "Optimal disturbance rejection for PI controller with constraints on relative delay margin". *ISA Transactions* **63**, pp. 103–111. ISSN: 0019-0578. DOI: https://doi.org/10.1016/j.isatra.2016.03.014.

Törngren, M. (1998). "Fundamentals of implementing real-time control applications in distributed computer systems". *Real-Time Systems* **14**:3, pp. 219–250. DOI: 10.1023/A:1007964222989.

Vankeerberghen, G., J. Hendrickx, and R. M. Jungers (2014). "JSR: a toolbox to compute the joint spectral radius". In: *17th International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, Berlin, Germany, pp. 151–156. ISBN: 9781450327329.

Vreman, N., A. Cervin, and M. Maggio (2021a). "Stability and performance analysis of control systems subject to bursts of deadline misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. DOI: 10.4230/LIPIcs.ECRTS.2021.15.

Vreman, N., A. Cervin, and M. Maggio (2021b). "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.15.

Vreman, N. and C. Mandrioli (2020). "Evaluation of Burst Failure Robustness of Control Systems in the Fog". In: *2nd Workshop on Fog Computing and the IoT (Fog-IoT)*. Vol. 80. OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-95977-144-3. DOI: 10.4230/OASIcs.Fog-IoT.2020.8.

Wang, W., P. Mishra, and A. Gordon-Ross (2012). "Dynamic cache reconfiguration for soft real-time systems". *ACM Trans. Embed. Comput. Syst.* **11**:2. ISSN: 1539-9087. DOI: 10.1145/2220336.2220340.

Wellstead, P. E., V. Chrimes, P. R. Fletcher, and A. J. R. R. Moody (1978). "The ball and beam control experiment". *The International Journal of Electrical Engineering & Education* **15**:1.

Xiong, J. and J. Lam (2007). "Stabilization of linear systems over networks with bounded packet loss". *Automatica* **43**:1, pp. 80–87. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2006.07.017.

Ying Zhang and Krishnendu Chakrabarty (2003). "Fault recovery based on checkpointing for hard real-time embedded systems". In: *18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 320–327. DOI: 10.1109/DFTVS.2003.1250127.

Zhang, W. and L. Yu (2010). "Stabilization of sampled-data control systems with control inputs missing". *IEEE Transactions on Automatic Control* **55**:2, pp. 447–452. DOI: 10.1109/TAC.2009.2036325.

Zhang, W., M. S. Branicky, and S. M. Phillips (2001). "Stability of networked control systems". *IEEE Control Systems Magazine* **21**:1, pp. 84–99. DOI: 10.1109/37.898794.

Paper III

`WeaklyHard.jl`: Scalable Analysis of Weakly-Hard Constraints

Nils Vreman Richard Pates Martina Maggio

Abstract

Weakly-hard models have been used to analyse real-time systems subject to patterns of deadline hits and misses. However, the tools that are available in the literature have a set of shortcomings. The analysis they offer is limited to a single weakly-hard constraint and to patterns that specify the number of misses, rather than the number of hits. Furthermore, the scalability of the tools is limited, effectively making it hard to address systems where deadline misses are really sporadic events. In this paper we present `WeaklyHard.jl`, a scalable tool to analyse a set of weakly hard constraints belonging to all the four types of weakly hard models. To achieve scalability, we exploit novel dominance relations between weakly-hard constraints, based on deadline hits. We provide experimental evidence of the tool's scalability, compared to the state-of-the-art for a single constraint, a thorough investigation of hit-based weakly-hard constraints, and a sensitivity analysis to constraint set parameters.

Originally published in IEEE 28th Real-Time and Embedded Technology and Applications Symposium (2022). The mathematical notation has been unified to match the remainder of the thesis. Reprinted with permission.

1. Introduction

A recent survey on the state of industrial practice in real-time systems showed that a significant fraction of real-time tasks are allowed to miss a finite number of deadlines [Åkesson et al., 2020]. The research community spent years defining and analysing models of tasks that can miss deadlines, from soft real-time systems [Buttazzo et al., 2005], to tasks with a skip-factor [Koren and Shasha, 1995], from calculating the miss ratio based on execution time probability distributions [Manolache et al., 2004], to approximating the deadline miss probability [Brüggen et al., 2018; Bozhko et al., 2021; Brüggen et al., 2021] for a given system.

One of such models in which tasks may miss deadlines is the weakly-hard task model [Bernat et al., 2001]. Weakly-hard tasks behave according to patterns of hit and missed deadlines that are (mainly) window-based. The originally proposed constraint models specifies alternatively (for a window of subsequent jobs): (i) the minimum number of deadlines that are hit, (ii) the minimum number of consecutive deadlines that are hit, (iii) the maximum number of deadlines that may be missed, or (iv) the maximum number of consecutive deadlines that may be missed. The third of these models – often called the (m,K) model – gained attention in the research community, generating results on scheduling algorithms [Hamdaoui and Ramanathan, 1995], real-time and schedulability analysis [Sun and Natale, 2017; Pazzaglia et al., 2021b; Hammadeh et al., 2017c], verification [Huang et al., 2019a; Behrouzian et al., 2020] and runtime monitoring [Wu et al., 2020] of constraint satisfaction, derivation of task model parameters [Xu et al., 2015], together with applications to domains like telecommunication [Ahrendts et al., 2018; Huang et al., 2019b] and control systems [Ramanathan, 1999; Pazzaglia et al., 2018; Vreman et al., 2021; Pazzaglia et al., 2021a]. The fourth model has also proved relevant to perform analyses of the stability of control systems [Maggio et al., 2020]. Furthermore, the relation between weakly-hard constraint types has been partially investigated [Tu et al., 2007; Wu et al., 2020]. However, this investigation remains partial as some of the constraints are not connected and their dominance (i.e., the comparison of how strictly does the task model constrain the task execution for different types of constraints) is not assessed.

The practical usefulness of weakly-hard models will remain limited, unless it is possible to build tools to enforce and monitor the satisfaction of weakly-hard constraints for execution platforms. Many real-time platforms offer the possibility to invoke "protected" task executions, ensuring that deadlines are met at the cost of increasing the execution cost. This is a very simple mechanism to secure that the weakly-hard constraint is satisfied in an execution platform. However, this requires writing monitoring code, that generates transition points to this protected execution mode when a constraint might otherwise be violated. Generating this code in a scalable way requires abstracting from the constraint and representing the execution of tasks with compact, but expressive, models.

To date, the literature has focused on the (m,K) constraint, neglecting the others,

despite their relevance in application domains such as control [Maggio et al., 2020; Linsenmayer and Allgower, 2017; Vreman et al., 2021]. As a result, the mentioned tools and models are not available for all the constraint types. This paper aims at both solving this problem and answering some open issues, namely: (i) guaranteeing consecutive deadline hits, and not only following patterns of deadline misses; and (ii) dealing with systems that satisfy multiple weakly-hard constraint simultaneously.

The first issue comes from the consideration that in practice it may be easier to guarantee that some prescribed job will hit their deadline rather than ensuring that the number of misses follows a given pattern. This is the case of the mentioned protected execution environment. As an example, mixed-criticality allows the scheduler to raise the criticality level and thus guarantee that the highly-critical tasks meet the corresponding deadlines [Burns and Davis, 2013]. We can treat the weakly-hard task as highly critical and raise the criticality level when a deadline hit must be enforced. Alternatively, we can increase the budget of a reservation-based scheduler [Casini et al., 2019]. Despite the fact that guaranteeing hits is often easier than enforcing miss patterns, the first two types of weakly hard tasks, that constrain the number of hits, have not been receiving much attention from the research community.

Furthermore, we would like to analyse tasks that satisfy multiple constraints simultaneously. Most analysis methods only take into account a single constraint, e.g., [Pazzaglia et al., 2018] or [Maggio et al., 2020] for the stability of control systems. In some cases, one of the two constraints *dominates* the other, meaning that satisfying the dominant constraint also guarantees the satisfaction of the dominated one. But this is not always the case. Consider for example two constraints λ_1 and λ_2, where λ_1 specifies that the task may miss a maximum of 2 deadlines in every window of 5 consecutive jobs, and λ_2 that it may miss a maximum of 3 deadlines in every window of 7 consecutive jobs. On the one hand the sequence 0011100, where 0 represents a deadline miss and 1 a deadline hit, satisfies λ_1 but fails λ_2, meaning that λ_2 does not dominate λ_1. On the other hand the sequence 0001111 satisfies λ_2 but fails λ_1, and so λ_1 does not dominate λ_2 either. If the analysis can only be conducted with a single constraint, the choice of which constraint is to be used is left to the practitioner, while it would be best to consider *both* constraints simultaneously.

Finally, we bring forward the question of *scalability*. Many of the research results, for example in the control domain [Pazzaglia et al., 2018; Linsenmayer and Allgower, 2017; Linsenmayer et al., 2021], use short windows. However, for practical applications it may be relevant to use a large window size, as done for example in the experimental analysis in [Behrouzian et al., 2020]. In fact, the original motivation behind the weakly-hard task model [Bernat et al., 2001] uses a practical example from the avionics domain in which a deadline may be missed 11 times in every consecutive 295 jobs. It seems reasonable that systems that are built and certified (for example in the automotive domain) would not experience many deadline

misses, and that using a short window size would lead to very conservative results.

To address these questions and empower researchers with a tool to apply their analysis techniques, this paper presents `WeaklyHard.jl`, a software library for weakly hard tasks that treats scalability as a first-class citizen. More precisely, the contributions of the paper are the following:

- We provide a theoretical contribution on the relation between weakly hard tasks that constrain the number of hits and the number of consecutive hits in a window (Section 3). This relation allows us to relate all the types of constraints with one another, and provide some ordering among them.

- We leverage an automata-based representation to describe the behaviour of a task subject to a weakly-hard constraint [Horssen et al., 2016; Linsenmayer et al., 2021]. In constrast to other approaches, our description exploits a mapping between a single transition in the automaton and a deadline (Section 4). This enables uses such as automatic generation of monitors to check weakly-hard constraint satisfaction on the fly.

- We extend the automaton to describe a task subject to a finite set of weakly-hard constraints (Section 3). In this way, we are able to address the analysis of systems that satisfy multiple constraints, possibly of different types, that do not dominate one another. As far as we know, this is the first paper that presents an analysis of a set of weakly-hard constraints.

We conduct an extensive performance evaluation campaign with a two-fold purpose (Section 5). First, we analyse the scalability of our library compared to the state of the art whenever possible, i.e., for single constraints. Second, we look at sets of constraints and perform a sensitivity analysis, to determine which parameters affect the execution time of the automaton construction for a set of constraints.

`WeaklyHard.jl` can be used for monitoring tasks subject to multiple weakly-hard constraints, analysing satisfaction sets, schedulability analysis, or connecting the weakly-hard model to applied fields like control theory. In particular, recent papers [Pazzaglia et al., 2018; Maggio et al., 2020; Vreman et al., 2021; Linsenmayer et al., 2021; Linsenmayer and Allgower, 2017] connected the weakly-hard model with control proofs considering stability and performance guarantees, and `WeaklyHard.jl` can generate general automata-based monitoring code ensuring the satisfaction of said properties.

2. Background and related work

In this work, we analyse a single real-time task. For the remainder of this paper, a real-time task τ is an entity composed of a sequence of jobs $(j_i)_{i \in \mathbb{N}_{\geq}}$, representing

code that is executed repeatedly on a given hardware platform (not necessarily according to any temporal pattern or periodicity). A task is characterised by its relative deadline d, representing the time after which each job should be completed.

The index i counts the job number. For a given job j_i, we denote with a_i its release time (the time in which the job becomes active in the hardware platform), and with f_i its completion time (the time in which the job terminates its execution). We also use d_i to represent the absolute deadline of the i-th job, meaning that $d_i = a_i + d$.

In general, a job can either complete its execution before its deadline or overrun it, resulting respectively in a deadline *hit* or *miss* (collectively denoted by the job's *outcome*).

DEFINITION 1—DEADLINE HIT
The i-th job of a task τ is said to hit its deadline if $f_i \leq d_i$.

DEFINITION 2—DEADLINE MISS
The i-th job of a task τ is said to miss its deadline if $f_i > d_i$.

The weakly-hard task model [Bernat et al., 2001; Bernat, 1998] provides guarantees on the sequence of outcomes of a real-time task via four constraints, each specifying how deadline misses and hits are interleaved for a window of $k \geq 1$ consecutive jobs.

DEFINITION 3—WEAKLY-HARD TASK
A weakly-hard task τ is a task that satisfies (at least) one of the following constraints:

(i) $\tau \vdash \binom{x}{k}$ *(`AnyHit`): in any window of k consecutive jobs, the minimum number of hits is x;*

(ii) $\tau \vdash \left\langle {x \atop k} \right\rangle$ *(`RowHit`): in any window of k consecutive jobs, the minimum number of consecutive hits is x;*

(iii) $\tau \vdash \overline{\binom{x}{k}}$ *(`AnyMiss`): in any window of k consecutive jobs, the maximum number of misses is x; and*

(iv) $\tau \vdash \overline{\left\langle {x \atop k} \right\rangle}$ *(`RowMiss`): in any window of k consecutive jobs, the maximum number of consecutive misses is x;*

for some values of $x \in \mathbb{N}_{>}$, $k \in \mathbb{N}_{>}$, where $x \leq k$. We use the $\vdash$ symbol to indicate that all the possible sequences of outcomes of τ satisfy the right hand side.

The types of constraints in Definition 3 have received different attention in the real-time systems literature. In particular, the `AnyMiss` constraint has been extensively studied, and is commonly addressed as the (m,K) weakly-hard task model [Hamdaoui and Ramanathan, 1995; Hammadeh et al., 2017b; Hammadeh et al., 2017a;

Sun and Natale, 2017; Ahrendts et al., 2018; Pazzaglia et al., 2018]. However, these constraints have been studied separately, while a task can simultaneously satisfy many, possibly of different types.

Exploiting different types of constraints – and possibly different parameters for the same type of constraint – leads to a better outcome for the analysis of the system. This follows from the space of possible sequences being pruned, thus allowing us to focus on proving that the real-time system behaves correctly in the relevant cases. In the following, we denote a set of L weakly-hard constraints with $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_L\}$. To characterise the possible sequences of outcomes that satisfy a constraint, we borrow some elementary concepts from language theory, in particular the *binary alphabet* [Hopcroft et al., 2006].

DEFINITION 4—ALPHABET Σ OF JOB OUTCOMES
We define the alphabet of job outcomes $\Sigma = \{0, 1\}$*, where* 0 *indicates a deadline miss and* 1 *represents a deadline hit.*

Using well-established notation, we denote the *character* $c_i \in \Sigma$ as the outcome of job j_i. A *word* w of length $|w| = N$ is a sequence of characters $w = \langle c_1, c_2, \ldots, c_N \rangle$ that specifies a sequence of consecutive job outcomes for a task. Without loss of generality, we assume that all words are preceded and followed only by hits. We denote the sub-word of a word w from index a to b with $w(a,b) = \langle c_a, c_{a+1}, \ldots, c_b \rangle$. Finally, Σ^N denotes the set of all possible words of length N.

With a slight abuse of notation, we use $w \vdash \lambda$ to indicate that the word w satisfies the constraint λ. Obtaining the set of words satisfying λ follows directly from the definitions of the alphabet and the constraint itself [Bernat et al., 2001; Bernat, 1998].

DEFINITION 5—SATISFACTION SET $\mathcal{S}_N(\lambda)$
The set of all length N words w, satisfying the weakly-hard constraint λ*, is denoted by* $\mathcal{S}_N(\lambda)$*. Formally,* $\mathcal{S}_N(\lambda) = \left\{ w \in \Sigma^N \,|\, w \vdash \lambda \right\}, N \geq 1$.

Trivially, all words in $\mathcal{S}_M(\lambda)$ are sub-words of words existing in $\mathcal{S}_N(\lambda)$, if $M \leq N$. To simplify notation we define the set containing all words of infinite length as $\mathcal{S}(\lambda) \equiv \mathcal{S}_\infty(\lambda)$.

Using satisfaction sets, it is possible to formally define a partial ordering between two constraints λ_i and λ_j. We denote the logical conjunction with $\wedge$ and the logical disjunction with $\vee$. The following notions of constraint *domination* and *equivalence* [Bernat et al., 2001; Bernat, 1998] are used extensively throughout the remainder of the paper (jointly denoted *constraint dominance*).

DEFINITION 6—CONSTRAINT DOMINATION
Given two arbitrary weakly-hard constraints λ_i *and* λ_j*,* λ_i *dominates* λ_j *(denoted* $\lambda_i \prec \lambda_j$*) if all words satisfying* λ_i *also satisfy* λ_j*, i.e.,* $\mathcal{S}(\lambda_i) \subset \mathcal{S}(\lambda_j)$*. Correspondingly,* $\lambda_i \preceq \lambda_j \Leftrightarrow \mathcal{S}(\lambda_i) \subseteq \mathcal{S}(\lambda_j)$.

DEFINITION 7—CONSTRAINT EQUIVALENCE
Given two arbitrary weakly-hard constraints λ_i and λ_j, λ_i is equivalent to λ_j if they respectively dominate each other. Formally, $\lambda_i \equiv \lambda_j \Leftrightarrow \lambda_i \preceq \lambda_j \wedge \lambda_j \preceq \lambda_i$. Two constraints are equivalent if they share the same satisfaction set, i.e., $\lambda_i \equiv \lambda_j \Leftrightarrow \mathcal{S}(\lambda_i) = \mathcal{S}(\lambda_j)$.

The notion of constraint dominance has attracted attention from different areas, and is still occasionally researched [Wu et al., 2020; Tu et al., 2007]. To provide dominance results, we first define the *weakest* and *hardest* constraints [Bernat et al., 2001; Bernat, 1998].

DEFINITION 8—WEAKEST CONSTRAINT $\underline{\lambda}$
The weakest constraint $\underline{\lambda}$ is defined as the constraint satisfied by any *word. Formally, $\mathcal{S}_N(\underline{\lambda}) = \Sigma^N, \forall N \in \mathbb{N}_>$.*

DEFINITION 9—HARDEST CONSTRAINT $\overline{\lambda}$
The hardest constraint $\overline{\lambda}$ is defined as the constraint satisfied solely *by the word containing all deadline hits. Formally, $\mathcal{S}_N(\overline{\lambda}) = \{1^N\}, \forall N \in \mathbb{N}_>$.*

Using these definitions, we now review known constraint dominance relations. We refer the reader to [Bernat, 1998] or any referenced paper for the corresponding proofs.

LEMMA 1—KNOWN EQUIVALENCE RELATIONS
The following equivalence relations hold:

(i) $\binom{x}{k} \equiv \overline{\binom{k-x}{k}}$, *an* **AnyHit** *constraint with x deadline hits in a window of k jobs is equivalent to an* **AnyMiss** *constraint with $k-x$ hits in a window of k jobs,*

(ii) $\overline{\langle {}^{x}_{k} \rangle} \equiv \overline{\langle x \rangle}$, $\forall k \geq 1$, *a* **RowMiss** *constraint is independent of the window size, i.e., it is equivalent to the same constraint with any k value,*

(iii) $\overline{\langle x \rangle} \equiv \overline{\binom{x}{x+1}}$, *a* **RowMiss** *constraint with x deadline misses is equivalent to an* **AnyMiss** *with x possible misses in a window of $x+1$ jobs [Maggio et al., 2020],*

(iv) $\langle {}^{1}_{k} \rangle \equiv \binom{1}{k}$, *(trivially) a* **RowHit** *constraint is equivalent to an* **AnyHit** *when looking at the same window length and a single deadline,*

(v) $\langle {}^{x}_{k} \rangle \equiv \overline{\lambda} \Leftrightarrow x > {}^{k}/_{2}$, *a* **RowHit** *constraint is equivalent to the hardest constraint when $x > {}^{k}/_{2}$.*

Using these equivalence relations, we can always translate **AnyMiss** and **RowMiss** constraints into a corresponding **AnyHit** constraint. However, there is no clear equivalence between **AnyHit** and **RowHit** constraints (beside the trivial

case of a single deadline and the same window length). Finding such a relation is important because it would allow us to treat sets of different types of constraints reducing the analysis to a single type and therefore improving efficiency. This motivates our formal analysis of the relation between hit-related constraints, presented in Section 3.

Denoting with $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ respectively the floor and ceiling operators, we can then define some domination relations.

LEMMA 2—KNOWN DOMINATION RELATIONS
The following domination relations hold:

(i) $\binom{x_1}{k_1} \preceq \binom{x_2}{k_2} \Leftrightarrow x_2 \leq \max\{a,b\}$*, where* $a = \lfloor \frac{k_2}{k_1} \rfloor x_1$ *and* $b = k_2 - \lceil \frac{k_2}{k_1} \rceil (k_1 - x_1)$*; the* **`AnyHit`** *constraint with parameters* x_1 *and* k_1 *dominates all* **`AnyHit`** *constraints with parameters* x_2 *and* k_2 *if and only if* $x_2 \leq \max\{a,b\}$ *with* a *and* b *defined as above.*

(ii) *For any two constraints* $\langle {}^{x_1}_{k_1} \rangle, \langle {}^{x_2}_{k_2} \rangle \not\equiv \overline{\lambda}$, $\langle {}^{x_1}_{k_1} \rangle \preceq \langle {}^{x_2}_{k_2} \rangle \Leftrightarrow (k_2 < k_1 \wedge k_2 \leq x_1 - \lceil \frac{k_1 - k_2}{2} \rceil) \vee (k_2 \geq k_1 \wedge x_2 \leq x_1)$*; this specifies the domination between two* **`RowHit`** *constraints depending on their constraint parameters.*

(iii) $\langle {}^{x_1}_{k} \rangle \preceq \binom{x_2}{k} \Rightarrow \{x_2 \leq 4x_1 - k - 2,\ x_2 \leq x_1,\ x_2 \geq 0\}$*; for a fixed and equal window* k*, if a* **`RowHit`** *constraint with consecutive deadlines hits* x_1 *dominates an* **`AnyHit`** *constraint with* x_2 *deadlines hits, then the indicated relation between the constraint parameters hold.*

(iv) $\overline{\langle x_1 \rangle} \preceq \overline{\langle x_2 \rangle} \Leftrightarrow x_1 \leq x_2$*; a* **`RowMiss`** *constraint with a lower number of deadline misses dominates a* **`RowMiss`** *with a higher number of deadline misses.*

(v) $\overline{\binom{x+p}{k+p}} \preceq \overline{\binom{x}{k}}$ *if* $p > 0$*;* **`AnyMiss`** *constraints can be dominated by other* **`AnyMiss`** *constraints when particular relations hold for values of their parameters [Tu et al., 2007].*

The ability to translate constraints into `AnyHit` equivalents makes Lemma 2(i) very powerful to compare different weakly hard constraints. Finally, Lemma 2(iii) is the only known result that relates the `RowHit` constraints with the other types. However, its applicability is limited to the case in which the two constraints share the same window size. From the presentation of the existing constraint dominance relations, we gather that there is an important piece missing to achieve a comprehensive weakly-hard analysis.

3. `AnyHit`, `RowHit`, and constraint sets

This section contains the theoretical contribution of the paper. In 3.1, we present some novel results on the relation between the `RowHit` and `AnyHit` constraints.

The results introduce the final theoretical pieces allowing us to relate all the weakly-hard constraint types to the AnyHit constraint, and thus to pave the way towards an efficient analysis implementation. In 3.2, we extend the theoretical results to handle sets of constraints, possibly containing constraints of different types.

3.1 Relating RowHit and AnyHit constraints

Our first theoretical contribution is the proof of a condition regarding the domination of a RowHit constraint over a AnyHit constraint, precisely

$$\left\langle {x_1 \atop k_1} \right\rangle \preceq \binom{x_2}{k_2} \Leftrightarrow x_2 \leq x_1 \lfloor k_2/p \rfloor + \max\{0, x_1 - p + (k_2 \bmod p)\}$$

with $p = k_1 - x_1 + 1$. The proof is based on restricting the AnyHit constraint's minimum number of hits in order to ensure that its satisfaction set includes the one of the RowHit constraint.

THEOREM 1—RowHit–AnyHit DOMINATION
Let $\mathcal{S}$ be the satisfaction set of the RowHit constraint $\lambda_1 = \left\langle {x_1 \atop k_1} \right\rangle$, and $k_2 \geq x_2$ be non-negative integers. Then the following are equivalent:

(i) Every sequence in $\mathcal{S}$ satisfies the AnyHit constraint $\binom{x_2}{k_2}$;

(ii) $x_2 \leq x_1 \lfloor k_2/p \rfloor + \max\{0, x_1 - p + (k_2 \bmod p)\}$, where $p = k_1 - x_1 + 1$.

Proof. We split the proof in two separate parts. First, we are going to prove that $\neg$*(ii)* $\Rightarrow \neg$*(i)*, and then we will prove that *(ii)* $\Rightarrow$ *(i)*, concluding the argument.

$\neg$*(ii)* $\Rightarrow \neg$*(i)*: Consider the binary sequence that alternates between x_1 consecutive 1's and $p - x_1$ consecutive 0's, where p is as in *(ii)*:

$$\bar{s} = \ldots \underbrace{1 \ldots 1}_{x_1} \overbrace{0 \ldots 0}^{p-x_1} \underbrace{1 \ldots 1}_{x_1} \overbrace{0 \ldots 0}^{p-x_1} \ldots \tag{1}$$

First observe that $\bar{s} \in \mathcal{S}$. Using the definitions of floor and modulo operator, for any integer value (including $p = k_1 + 1$) we can rewrite k_2 as $k_2 = \lfloor k_2/p \rfloor + (k_2 \bmod p)$. From the definition of sequence $\bar{s}$ in Equation (1), $\bar{s}$ certainly contains a sub-word of length k_2 with

$$x_1 \lfloor k_2/p \rfloor + \max\{0, x_1 - p + (k_2 \bmod p)\}$$

1's. If the inequality in *(ii)* does not hold, then $\bar{s}$ does not satisfy the AnyHit constraint $\lambda_2 = \binom{x_2}{k_2}$ (the sub-word of length k_2 above would contain fewer than x_2 1s).

(ii) $\Rightarrow$ *(i)*: Let s be any sequence in $\mathcal{S}$. Now let s' be equal to s, except that every maximal sub-word of 1s with fewer than x_1 elements has been replaced with a sub-word of zeros:

$$s'_i = \begin{cases} 1 & \text{if } s_i \text{ is part of a sub-word of at least } x_1 \text{ 1s,} \\ 0 & \text{otherwise.} \end{cases}$$

First observe that $s \in \mathcal{S}$ implies $s' \in \mathcal{S}$. This is because maximal sub-words of 1s with fewer than x_1 elements do not contribute to the satisfaction of a `RowHit` constraint (from the perspective of this constraint, such sub-words may as well be zeros). Also note that if s' satisfies an `AnyHit` constraint, so does s. This is because s can be obtained from s' by flipping 0s to 1s, which cannot lead to a violation of an `AnyHit` constraint. Therefore, it is sufficient to show that if *(ii)* holds, any such s' satisfies the `AnyHit` constraint in *(i)*. By construction, s' alternates between sub-words of 1's with at least x_1 elements, and sub-words of zeros of at most $p - x_1$ elements

$$s' = \dots \underbrace{1 \dots 1}_{\geq x_1} \overbrace{0 \dots 0}^{\leq p - x_1} \underbrace{1 \dots 1}_{\geq x_1} \overbrace{0 \dots 0}^{\leq p - x_1} \dots$$

It then follows that every sub-word of length k_2 in s' has at least as many 1's as every sub-word of length k_2 in the sequence $\bar{s}$ from (1). Since $\bar{s}$ satisfies the `AnyHit` constraint, so does s', and therefore so does every $s \in \mathcal{S}$ as required. □

The second theoretical contribution of the paper is the proof of a condition regarding the domination of an `AnyHit` constraint over a `RowHit` constraint, specifically

$$\binom{x_1}{k_1} \preceq \left\langle \begin{matrix} x_2 \\ k_2 \end{matrix} \right\rangle \Leftrightarrow x_2 \leq \min\left\{ \left\lfloor k_2 / (z_1 + 1) \right\rfloor, \left\lceil x_1 / z_1 \right\rceil \right\}$$

where $z_1 = k_1 - x_1$.

THEOREM 2—`AnyHit`–`RowHit` DOMINATION
Let $\mathcal{S}$ be the satisfaction set of the `AnyHit` *constraint $\binom{x_1}{k_1}$, and $k_2 \geq x_2$ be non-negative integers. Then the following are equivalent:*

(i) Every sequence in $\mathcal{S}$ satisfies the `RowHit` *constraint $\left\langle \begin{matrix} x_2 \\ k_2 \end{matrix} \right\rangle$;*

(ii) $x_2 \leq \min\{\lfloor k_2/(z_1+1) \rfloor, \lceil x_1/z_1 \rceil\}$, where $z_1 = k_1 - x_1$.

Proof. We split the proof in two separate parts. First, we are going to prove that $\neg$*(ii)* $\Rightarrow \neg$*(i)*, and then we will prove that $\neg$*(i)* $\Rightarrow \neg$*(ii)*, concluding the argument.

$\neg$*(ii)* $\Rightarrow \neg$*(i)*: We split the proof into three cases.

Case 1: $0 < k_2 \leq z_1$. Let $\bar{s} = \dots s_d s_d s_d \dots$ (i.e. the sequence constructed by repeating the sub-word s_d), where

$$s_d = \underbrace{1 \dots 1}_{x_1} \overbrace{0 \dots 0}^{z_1}.$$

Observe that $\bar{s} \in \mathcal{S}$. Since $\lfloor k_2 / (z_1 + 1) \rfloor = 0$, $\neg$*(ii)* implies that $x_2 > 0$. This implies $\neg$*(i)* because $\bar{s}$ contains at least k_2 consecutive 0s, and therefore cannot satisfy the `RowHit` constraint $\left\langle \begin{matrix} x_2 \\ k_2 \end{matrix} \right\rangle$.

Case 2: $k_2 > z_1 \wedge \lceil x_1/z_1 \rceil \geq \lfloor k_2/(z_1+1) \rfloor$. Let s_d be a sequence of length k_2 consisting of $k_2 - z_1$ 1s and z_1 0s, with the 1s arranged into z_1+1 sub-words

$$s_d = \underbrace{1\ldots1}_{l_1}0\underbrace{1\ldots1}_{l_2}0\ldots0\underbrace{1\ldots1}_{l_{z_1+1}},$$

where the lengths of the sub-words l_k satisfy

$$l_k \in \left\{ \left\lfloor \frac{k_2 - z_1}{z_1+1} \right\rfloor, \left\lceil \frac{k_2 - z_1}{z_1+1} \right\rceil \right\}.$$

Let $\bar{s} = \ldots 111 s_d 111 \ldots$ (i.e. a sequence of all 1s except for a single sub-word s_d). Since this sequence contains only z_1 0s, $\bar{s} \in \mathcal{S}$. The conclusion now follows since

$$\left\lceil \frac{k_2 - z_1}{z_1+1} \right\rceil = \left\lfloor \frac{k_2 - z_1 - 1}{z_1+1} \right\rfloor + 1 = \left\lfloor \frac{k_2}{z_1+1} \right\rfloor,$$

and so if $x_2 > \lfloor k_2/(z_1+1) \rfloor$, then this $\bar{s}$ does not satisfy the `RowHit` constraint $\left\langle {x_2 \atop k_2} \right\rangle$.

Case 3: $k_2 > z_1 \wedge \lceil x_1/z_1 \rceil < \lfloor k_2/(z_1+1) \rfloor$. Let s_d be a sequence of length k_1 consisting of x_1 1s and z_1 0s, with the 1s arranged into z_1 sub-words

$$s_d = \underbrace{1\ldots1}_{l_1}0\underbrace{1\ldots1}_{l_2}0\ldots0\underbrace{1\ldots1}_{l_{z_1}}0,$$

where the lengths of the sub-words l_k satisfy $l_k \in \{\lfloor x_1/z_1 \rfloor, \lceil x_1/z_1 \rceil\}$. Let $\bar{s} = \ldots s_d s_d s_d \ldots$ (i.e. the sequence constructed by repeating the sub-word s_d). Observe that every sub-word of length k_1 in $\bar{s}$ contains exactly x_1 1s, and therefore $\bar{s} \in \mathcal{S}$. Observe also that $\bar{s}$ contains no sub-words of more than $\lceil x_1/z_1 \rceil$ consecutive 1s, and therefore if $x_2 > \lceil x_1/z_1 \rceil$, $\bar{s}$ does not satisfy the `RowHit` constraint $\left\langle {x_2 \atop k_2} \right\rangle$.

¬(i) ⇒ ¬(ii): Under the hypothesis of *¬(i)*, there exists a sequence $s \in \mathcal{S}$ such that s does not satisfy the `RowHit` constraint $\left\langle {x_2 \atop k_2} \right\rangle$.

Let s' be the sequence obtained from s by removing all 0s from the start of s, and then replacing all sub-words of 0s with length greater than one with a single 0 (for example, if $s = 011001010001\ldots$, then $s' = 11010101\ldots$. Clearly $s' \in \mathcal{S}$ since this process only removes 0s, and s' also does not satisfy the `RowHit` constraint. Consider now the sub-word s_d formed from the first k_2 elements of s'.[1] This sub-word will take the form

$$s_d = \begin{cases} \underbrace{1\ldots1}_{l_1}0\underbrace{1\ldots1}_{l_2}0\ldots0\underbrace{1\ldots1}_{l_n}, & \text{or} \\ \underbrace{1\ldots1}_{l_1}0\underbrace{1\ldots1}_{l_2}0\ldots0\underbrace{1\ldots1}_{l_n}0, \end{cases}$$

[1] Strictly speaking if s is too short, then the sequence s' resulting from this process might have length less than $\min\{k_1, k_2\}$ which would mean that the statement $s' \in \mathcal{S}$ is ill defined. In this case 0s should only be removed until s' has length $\min\{k_1, k_2\}$. This will still result in a sequence that satisfies the `AnyHit` constraint but violates the `RowHit` constraint. All the given arguments remain valid for such an s', since they only depend on inequalities based on the number of 0s in particular sub-words of length k_1 as guaranteed by the `AnyHit` constraint (note in *Case 1* it is perfectly valid for $l_1 = 0$).

depending on whether the final element is 0 or 1. Note that the lengths of the sub-words of 1s satisfy $0 \leq l_k < x_2$. We will now show that the existence of such a sub-word implies $\neg$*(ii)* by considering two cases.

Case 1: $k_2 \leq k_1 + l_1$. In this case the sub-word s_d contains at most z_1 0s, and so $n \leq z_1 + 1$. The pigeonhole principle then demonstrates that there must be an integer $1 \leq k \leq n$ such that

$$l_k \geq \left\lceil \frac{k_2 - z_1}{n} \right\rceil.$$

To see this, note that s_d has at least $k_2 - z_1$ 1s, and these must be allocated into n pigeonholes corresponding to the n sub-words of 1s. This implies that

$$x_2 > l_k \geq \left\lceil \frac{k_2 - z_1}{n} \right\rceil \geq \left\lceil \frac{k_2 - z_1}{z_1 + 1} \right\rceil = \left\lfloor \frac{k_2}{z_1 + 1} \right\rfloor$$

and $\lfloor k_2/(z_1+1) \rfloor \geq \min\{\lfloor k_2/(z_1+1) \rfloor, \lceil x_1/z_1 \rceil\}$ as required.

Case 2: $k_2 > k_1 + l_1$. Let s'_d denote the sub-word obtained by removing the first l_1 elements of s_d, and also removing elements from the end of s_d, until s'_d has length k_1. This sub-word takes the form

$$s'_d = \begin{cases} 0\underbrace{1\ldots1}_{l_2}0\underbrace{1\ldots1}_{l_3}0\ldots0\underbrace{1\ldots1}_{l_{m+1}}, & \text{or} \\ 0\underbrace{1\ldots1}_{l_2}0\underbrace{1\ldots1}_{l_3}0\ldots0\underbrace{1\ldots1}_{l_{m+1}}0, \end{cases}$$

depending on whether the final element is 0 or 1. Since s'_d satisfies the `AnyHit` constraint $\binom{x_1}{k_1}$, it contains at most z_1 zeros, and so $m \leq z_1$. Therefore, in this case the pigeonhole principle implies that at least one of the lengths l_k must satisfy

$$l_k \geq \left\lceil \frac{x_1}{m} \right\rceil \geq \left\lceil \frac{x_1}{z_1} \right\rceil.$$

This implies $x_2 > l_k \geq \lceil x_1/z_1 \rceil \geq \min\{\lfloor k_2/(z_1+1) \rfloor, \lceil x_1/z_1 \rceil\}$ as required. □

The two theorems above complete the relation graph between the different types of weakly-hard constraints. Now that we have a complete picture, we can start investigating sets Λ of L constraints, $\Lambda = \{\lambda_1, \ldots, \lambda_L\}$.

3.2 Handling sets of weakly-hard constraints Λ

We extend the theory to the case in which τ is subject to an arbitrary set of constraints of the form presented in Definition 3. First, we extend the satisfaction from Definition 5 and obtain

$$\mathcal{S}_N(\Lambda) = \bigcap_{\lambda \in \Lambda} \mathcal{S}_N(\lambda) \tag{2}$$

where $\bigcap$ is the generalised intersection. We use $\tau \vdash \Lambda$ to denote that τ satisfies all the constraints in the set Λ. This implies that each word $w \in \mathcal{S}_N(\Lambda)$ must belong to the satisfaction set of all the constraints in Λ. Trivially, Equation (2) allows us to extended Definitions 5 and 6 to define constraint dominance for sets of constraints.

Constraint dominance significantly reduces the problem complexity when working with sets of weakly-hard constraints, Λ. If the constraint set supports different types of weakly-hard constraints, it can be beneficial to find an equivalent set of constraints with minimal cardinality.

To minimise the number of constraints in the problem formulation, the constraint dominance is utilised in order to find the minimal cardinality, equivalent subset. Utilising the comprehensive picture the theorems provide, we propose the notion of a *dominant set*, thus simplifying the analysis of weakly-hard systems subject to multiple constraints.

DEFINITION 10—DOMINANT SET
The dominant set Λ^* *of a set of weakly-hard constraints* Λ *is defined as the smallest cardinality subset of* Λ *representing an equivalent set of constraints. Formally,* $\Lambda^* \subseteq \Lambda$ *where*

(i) $\lambda_i, \lambda_j \in \Lambda^* \;\Rightarrow\; \lambda_i \not\equiv \lambda_j,\ \forall i \neq j,$

(ii) $\lambda_i, \lambda_j \in \Lambda^* \;\Rightarrow\; \lambda_i \not\preceq \lambda_j,\ \forall i \neq j,$

(iii) $\lambda_i \in \Lambda \setminus \Lambda^* \;\Rightarrow\; \exists \lambda_j \in \Lambda^*\ s.t.\ \lambda_j \preceq \lambda_i.$

From Definition 6, a weakly-hard constraint λ_i dominates λ_j if and only if $\mathcal{S}(\lambda_i) \subseteq \mathcal{S}(\lambda_j)$. Thus, excluding all the dominated constraints from Λ does not change the resulting satisfaction set. The equivalence between the constraint set and its dominant set is trivial considering the respective satisfaction sets:

$$\mathcal{S}(\Lambda^*) = \bigcap_{\lambda \in \Lambda^*} \mathcal{S}(\lambda) = \bigcap_{\lambda \in \Lambda} \mathcal{S}(\lambda) = \mathcal{S}(\Lambda).$$

In the following section, we present our tool, `WeaklyHard.jl`, and use the theorems presented in this section and the dominance between constraints to simplify the analysis of sets of weakly-hard constraints.

4. `WeaklyHard.jl`

In this section we introduce `WeaklyHard.jl`[2], a scalable tool for analysing (sets of) weakly-hard constraints of different types. The tool facilitates the analysis of weakly-hard tasks providing functions to:

[2] `https://github.com/NilsVreman/WeaklyHard.jl`

(i) compare two arbitrary weakly-hard constraints or two sets of weakly-hard constraints, obtaining answers about their dominance,

(ii) translate a weakly-hard constraint or a set of weakly-hard constraints into a corresponding automaton, that represents all the sequences that belong to the satisfaction set of the set of constraints,

(iii) produce *all* sequences of arbitrary length that satisfy a set of weakly-hard constraints, i.e., the satisfaction set.

We distribute `WeaklyHard.jl` as an open-source package, written in the Julia programming language [Bezanson et al., 2017]. Julia is a scripting language with Just-In-Time compilation. The language design is centered upon two core concepts: type-stability and function specialisation through multiple-dispatch. The type-stable compilation provides an implementation that is close to the hardware, resulting in efficient code execution. Multiple-dispatching allows us to write a user-friendly code library. Additionally, Julia's built in package manager simplifies the distribution of non-proprietary packages.

A task subject to any weakly-hard constraint (from Definition 3) can be represented using an automaton. Automata have been used in the analysis of networked systems [Huang et al., 2019b; Osch and Smolka, 2001], schedulability [Zeng and Di Natale, 2012; Fersman et al., 2002; Fersman et al., 2007], and control systems [Linsenmayer and Allgower, 2017; Linsenmayer et al., 2021; Pazzaglia et al., 2018; Horssen et al., 2016]. In this paper, we decided to constrain the automaton structure, thinking about the possible use of the automaton, e.g., generating a monitor to check whether a constraint is satisfied. In our representation, vertices encode the task's state, i.e., the relevant suffix of the sequence of job outcomes. Similarly, edges are associated with a *feasible* outcome (hit or miss) and encode the transitions from one state to another. Feasibility here refers to the fact that deadline misses are not allowed if the constraint would not permit them. The outcome sequences acquired from *all* random walks in the automaton correspond to the satisfaction set of the weakly-hard constraint represented by the automaton.

Due to their combinatorial nature, weakly-hard systems are inherently complicated to analyse. Their complexity becomes apparent in the size of the automaton, and evidently grows when the window length of the constraint increases. In the following, we present a scalable approach for generating automata representations of weakly-hard constraints.

4.1 Weakly-hard constraints as automata

Suppose that $\tau \vdash \lambda$. We use $\mathcal{G}_\lambda = (V_\lambda, E_\lambda)$ to indicate the directed labeled graph $\mathcal{G}_\lambda$ corresponding to the automaton representation of τ. Here, V_λ represents the set of *vertices* in the graph and E_λ represents the directed *edges* between vertices (also denoted *transitions*). Each vertex $v_i \in V_\lambda$ represents a word $w_i \in \mathcal{S}(\lambda)$. With a slight notational abuse, vertices v_i will occasionally (when evident from context)

be treated as the word they represent, w_i. The transition $e_{i,j} \in E_\lambda$ corresponds to a tuple $e_{i,j} = (v_i, v_j, c_{i,j})$, where the vertex pair $v_i, v_j \in V_\lambda$ denotes the tail and head of the transition, and the character $c_{i,j} \in \Sigma$ corresponds to the transition's label. A transition $e_{i,j}$ is feasible if and only if the concatenation of the character $c_{i,j}$ to the word w_i satisfies λ. Formally:

$$e_{i,j} \in E_\lambda \Leftrightarrow \left\langle w_i\left(2, |w_i|\right), c_{i,j} \right\rangle = w_j \vdash \lambda.$$

Finally, for two vertices $v_i, v_j \in V_\lambda$ we say that v_j is a direct successor of v_i if there exists a transition $e_{i,j} \in E_\lambda$. Without loss of generality, we will assume that each vertex $v_i \in V_\lambda$ can have at most two direct successors with distinct transition outcomes, i.e., one successor v_{j_1} through $e_{i,j_1} = (v_i, v_{j_1}, 1)$ and (if permissible) one successor v_{j_0} through $e_{i,j_0} = \left(v_i, v_{j_0}, 0\right)$.

4.2 Automaton construction

The naïve approach of constructing the automaton $\mathcal{G}_\lambda$ is both time consuming and memory intensive (including $|\mathcal{S}_k(\lambda)|$ vertices, where k is the window length of λ). In order to improve performance and scalability, we include the following optimisations:

(i) representing words as bit strings,

(ii) minimising the automata size by combining equivalent vertices during the automata generation, and

(iii) representing large sets of constraints with their dominant subset.

Support for bit string operations (like shifting) is essential for efficient sequence management. Logical and bitwise operations are directly supported by all processors, thus they are highly optimised and require a minimal amount of instruction cycles. We use the following notation: & is the *bitwise and*, | is the *bitwise or*, and $\ll$ is the *logical left-shift*.

Each word $w \in \mathcal{S}(\lambda)$ is a sequence of outcomes and can therefore be interpreted as a string of bits – recall that an outcome is a character in $\Sigma = \{0, 1\}$. The rightmost character in w is the outcome of the last job, e.g., $w = 001$ implies that the last deadline was hit, but the two previous ones were missed. Assuming that the task τ experienced the outcomes w and the next outcome is $c \in \Sigma$, then the new sequence of outcomes is $w' = (w \ll 1) \,|\, c$.

The size of the naïve automaton can be reduced substantially by combining vertices that would otherwise result in language-equivalent states [Hopcroft et al., 2006]. Two vertices $v_{i_1}, v_{i_2} \in V_\lambda$ are considered equivalent if they share the same direct successors with the same transition outcomes. As an example, consider the `AnyHit` constraint $\lambda = \binom{1}{2}$. Trivially there are only three feasible vertices in the naïve automaton, since there are $2^k = 4$ words in Σ^k and $w = 00$ is infeasible. The

Algorithm 1 Generation of the minimal automaton representation $\mathcal{G}_\lambda$ corresponding to a weakly-hard constraint λ.

```
1:  procedure BUILDAUTOMATON(λ)
2:      V_λ ← {v_1 = (1 ≪ n) − 1}
3:      E_λ ← ∅, Q = {v_1}
4:      while Q ≠ ∅ do
5:          v_i ← pop(Q)
6:          v_{j_0} ← compact(λ, (v_i ≪ 1) | 0)
7:          v_{j_1} ← compact(λ, (v_i ≪ 1) | 1)
8:          if v_{j_0} ⊢ λ then
9:              if v_{j_0} ∉ V_λ then
10:                 V_λ ← V_λ ∪ {v_{j_0}}
11:                 Q ← Q ∪ {v_{j_0}}
12:             E_λ ← E_λ ∪ {e_{i,j_0} = (v_i, v_{j_0}, 0)}
13:         if v_{j_1} ∉ V_λ then
14:             V_λ ← V_λ ∪ {v_{j_1}}
15:             Q ← Q ∪ {v_{j_1}}
16:         E_λ ← E_λ ∪ {e_{i,j_1} = (v_i, v_{j_1}, 1)}
        return G_λ = (V_λ, E_λ)
```

words $w_1 = 11$ and $w_2 = 01$ are equivalent since they share the same direct successors with the same transition outcomes, i.e., $(w_1 \ll 1) \,|\, 0 = (w_2 \ll 1) \,|\, 0$ and $(w_1 \ll 1) \,|\, 1 = (w_2 \ll 1) \,|\, 1$, considering the window $k = 2$. Intuitively, the fact that it is possible to combine vertices comes from the realisation that a task's history, prior to the last k job outcomes, is irrelevant. Combining the equivalent vertices results in a new vertex representing the word $w = w_1 \,\&\, w_2$.

Finally, for sets of weakly-hard constraints Λ we construct the automaton $\mathcal{G}_{\Lambda^*}$ for the dominant set $\Lambda^* \subseteq \Lambda$. Since $\mathcal{S}(\Lambda^*) = \mathcal{S}(\Lambda)$, it also follows that $\mathcal{G}_{\Lambda^*} \equiv \mathcal{G}_\Lambda$.

We generate the minimal automaton $\mathcal{G}_\lambda$ as presented in Algorithm 1. The automaton is initialised with a single vertex corresponding to the word $w_1 = 1^n$, $v_1 = (1 \ll n) - 1$. Here, n is the smallest number of hits required in a window to meet the constraint λ, e.g., $n = 1$ for $\lambda = \overline{\langle 3 \rangle}$ or $n = 2$ for $\left\langle {2 \atop 5} \right\rangle$. As long as there exists uninitialised vertices v_i, its successors v_{j_0} and v_{j_1} are created and passed through a function in order to *compact* them. This step reduces the new word to the minimal, equivalent word that would still satisfy λ. In particular, if either $(v_i \ll 1) \,|\, 0$ or $(v_i \ll 1) \,|\, 1$ return an existing vertex v_{i_0} or v_{i_1}, then v_{j_0} and v_{j_1} are reduced to the corresponding existing one. If the resulting words would satisfy λ, they are properly added to the automaton. Note that it is only required to verify that the successor following a deadline miss satisfy the constraint.

Notice that minimality comes from the fact that we include a vertex in $\mathcal{G}_\lambda$ only if there exists no other vertex that represents the same sequence. In fact, each new

vertex added to the automaton represents a feasible sequence that no other vertex is already encoding. If a potential new vertex represents a sequence that is *equivalent* to another existing vertex, the algorithm connects the existing vertex instead of creating a new one.

4.3 Scalable automata generation

Intuitively, the time required for generating an automaton is directly correlated to its size, i.e., more vertices lead to a larger exploration time and hence to a larger automaton-construction time. Additionally, the automata-based representation can be used in embedded devices, e.g., to monitor the satisfaction of a constraint. Thus, space and memory requirements create a clear need for the automaton to be minimal.

We provide a brief discussion on the minimum number of vertices needed to express the automaton corresponding to the weakly hard constraints presented in Definition 3. The structure of the minimal automaton depends on the type of constraint. For example, to describe an `AnyHit` constraint $\binom{x_{ah}}{k_{ah}}$ we need to keep track of the number and the position of the deadline hits we encountered in the past k_{ah} outcomes, giving us a number of vertices that corresponds to the binomial coefficient k_{ah} *choose* x_{ah}. The `AnyMiss` constraint can be reduced to the `AnyHit` constraint and hence we easily obtain the number of its vertices. For the `RowMiss` constraint, the number of vertices is also obvious, as we need to count the number of consecutive deadlines that have been missed, and return to the initial state as soon as the following outcome is a hit. Denoting with $\mathrm{s}(\lambda)$ the function that counts the number of vertices of the minimal automaton corresponding to the constraint λ, we obtain:

$$\texttt{AnyHit}: \quad \lambda_{ah} = \binom{x_{ah}}{k_{ah}} \Rightarrow \mathrm{s}(\lambda_{ah}) = \frac{k_{ah}!}{x_{ah}!\,(k_{ah}-x_{ah})!}$$

$$\texttt{AnyMiss}: \quad \lambda_{am} = \overline{\binom{x_{am}}{k_{am}}} \Rightarrow \mathrm{s}(\lambda_{am}) = \frac{k_{am}!}{x_{am}!\,(k_{am}-x_{am})!}$$

$$\texttt{RowMiss}: \quad \lambda_{rm} = \overline{\left\langle \begin{matrix} x_{rm} \\ k_{rm} \end{matrix} \right\rangle} \Rightarrow \mathrm{s}(\lambda_{rm}) = x_{rm}+1$$

e.g., the minimal automaton for the `AnyMiss` constraint $\overline{\binom{5}{20}}$ includes 15 504 vertices.

The `RowHit` constraint, $\left\langle \begin{matrix} x_{rh} \\ k_{rh} \end{matrix} \right\rangle$ is more interesting. When $k_{rh} < 2x_{rh}$, the constraint reduces to the hardest constraint $\overline{\lambda}$, hence the automaton has a single vertex. If $k_{rh} = 2x_{rh}$, it is possible to have a single deadline miss, that can only appear before a sequence of x_{rh} has been recorded, hence the corresponding automaton has $x_{rh}+1$ vertices. If $k_{rh} = 2x_{rh}+1$, the number of vertices of the automaton are $x_{rh}+2$

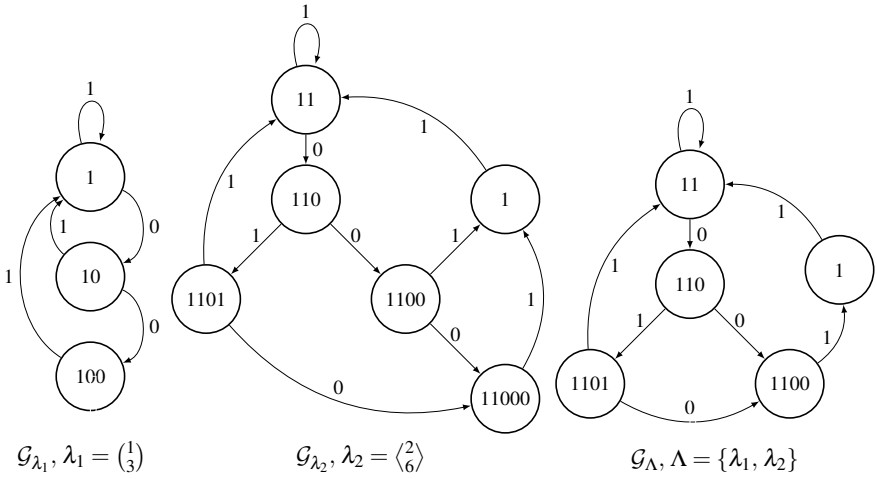


Figure 1. Minimal automata $\mathcal{G}_{\lambda_1}$, $\mathcal{G}_{\lambda_2}$, and $\mathcal{G}_{\Lambda}$ representing respectively λ_1, λ_2, and $\Lambda = \{\lambda_1, \lambda_2\}$ from the Example in Section 4.4.

and subsequent values can be found using recursion. Specifically,

$$\texttt{RowHit}: \lambda_{rh} = \left\langle {}^{x_{rh}}_{k_{rh}} \right\rangle \Rightarrow \mathrm{s}(\lambda_{rh}) =$$

$$\begin{cases} 1 & k_{rh} < 2x_{rh} \\ x_{rh}+1 & k_{rh} = 2x_{rh} \\ x_{rh}+2 & k_{rh} = 2x_{rh}+1 \\ 2\,\mathrm{s}(\langle {}^{x_{rh}}_{k_{rh}-1} \rangle) - \mathrm{s}(\langle {}^{x_{rh}}_{k_{rh}-2} \rangle) + 1 & 2x_{rh}+1 < k_{rh} < 3x_{rh} \\ \mathrm{s}(\langle {}^{x_{rh}}_{k_{rh}-1} \rangle) + x_{rh} & k_{rh} \geq 3x_{rh}. \end{cases}$$

In contrast to the `AnyHit` or `AnyMiss` constraints, the size of the minimal automaton corresponding to the `RowHit` constraint is linear in the window length k_{rh} in stationarity, i.e., when $k_{rh} \geq 3x_{rh}$. The linearity property also holds for the `RowMiss` constraint. Intuitively, since the size of the minimal automaton is directly correlated to the scalability, `RowHit` and `RowMiss` constraints are preferred for large problems.

4.4 Example

We now provide an example to illustrate how the automata differ between constraint types. In particular, we focus on `AnyHit` and `RowHit` constraints, that have been the subject of our theoretical investigation.

Given the two weakly-hard constraints $\lambda_1 = \binom{1}{3}$ and $\lambda_2 = \langle {}^{2}_{6} \rangle$, we apply Theorems 1 and 2 and confirm that there is no partial ordering between the constraints,

i.e. $\lambda_1 \not\preceq \lambda_2$ and $\lambda_2 \not\preceq \lambda_1$. Following the steps in Algorithm 1, we generate the *minimal* automaton representations of the two constraints, i.e., $\mathcal{G}_{\lambda_1}$ and $\mathcal{G}_{\lambda_2}$. The automaton representing the constraint set $\Lambda = \{\lambda_1, \lambda_2\}$, i.e., $\mathcal{G}_\Lambda$, is also generated and subsequently minimised. The results are shown in Figure 1, where the leftmost, middle, and rightmost automata correspond respectively to $\mathcal{G}_{\lambda_1}$, $\mathcal{G}_{\lambda_2}$, and $\mathcal{G}_\Lambda$.

One of the most important novelties presented in this paper is the possibility to analyse weakly-hard constraint *sets* containing *all* the weakly-hard constraints types from Definition 3. Prior work proposed alternative solutions to the automaton generation problem, handling either a specific type of constraint [Horssen et al., 2016], or a separate solution for each individual constraint type [Linsenmayer and Allgower, 2017]. Our aim is to switch the focus to the applicability and scalability of the constraint representation, and hence substitute `AnyHit` and `AnyMiss` with `RowHit` and `RowMiss` whenever possible. Being able to analyse sets of constraints in a scalable way brings us one step closer to the analysis of real systems, in which window lengths are quite large. Additionally, for real systems it is often easier to constrain hits (e.g., via execution in a protected environment without interference) rather than the maximum number or the pattern of deadline misses.

4.5 `WeaklyHard.jl` functionality

The most relevant functions provided by `WeaklyHard.jl` are summarised in Table 1.[3] In addition to the automata generation, the toolbox provides functions to compare constraints and obtain answers about their dominance and equivalence, to reduce a set of constraints to their dominant subset, and to generate sequences of arbitrary length satisfying sets of weakly-hard constraints. We also included a function that generate the satisfaction set $\mathcal{S}_N(\Lambda)$ from an automaton $\mathcal{G}_\Lambda$. In addition to the functions presented in Table 1, additional functions are included as syntactic sugar for a better user experience.

5. Experimental evaluation

We evaluate here the performance of `WeaklyHard.jl`.[4] First, we assess the scalability of the automaton generation, comparing `WeaklyHard.jl` with the state-of-the-art `WHRTgraph` [Linsenmayer and Allgower, 2017; Linsenmayer et al., 2021]. Then, we conduct a sensitivity analysis of `WeaklyHard.jl` to determine which parameters affect the execution time for the automata generation in cases that cannot be handled with other tools, e.g., sets of weakly-hard constraints. We provide results on how the type of constraints, maximum window length, and constraint set

[3] The package includes a README file that guides the user through the setup of the package and provides simple usage examples. The only prerequisite is the Julia interpreter and compiler, available at `https://julialang.org`.

[4] All the reported experiments ran on an Intel Xeon E5-2620 v3 @ 2.40GHz CPU with 126GB RAM memory.

Table 1. Functions offered by `WeaklyHard.jl`.

Function	Description
`AnyHitConstraint(x, k)`	Defines a constraint $\lambda = \binom{x}{k}$
`AnyMissConstraint(x, k)`	Defines a constraint $\lambda = \overline{\binom{x}{k}}$
`RowHitConstraint(x, k)`	Defines a constraint $\lambda = \langle {x \atop k} \rangle$
`RowMissConstraint(x)`	Defines a constraint $\lambda = \overline{\langle x \rangle}$
`is_satisfied(Lambda, w)`	Returns **true** if $w \vdash \Lambda$, i.e., if the word w satisfies all the constraints in Λ, and **false** otherwise (note: can be invoked also passing a single constraint λ as parameter)
`is_dominant(lambda1, lambda2)`	Returns **true** if $\lambda_1 \preceq \lambda_2$ and **false** otherwise
`is_equivalent(lambda1, lambda2)`	Returns **true** if $\lambda_1 \equiv \lambda_2$ and **false** otherwise
`dominant_set(Lambda)`	Returns $\Lambda^* \subseteq \Lambda$
`build_automaton(Lambda)`	Returns the automaton $\mathcal{G}_\Lambda$ (note: can be invoked also passing a single constraint λ as parameter)
`minimize_automaton!(G)`	Returns the minimal representation of $\mathcal{G}_\Lambda$ (note: changes $\mathcal{G}_\Lambda$)
`random_sequence(G, N)`	Returns a word w, $\lvert w \rvert = N$ obtained through an N-step random walk in $\mathcal{G}_\Lambda$
`all_sequences(G, N)`	Returns the satisfaction set $\mathcal{S}_N(\Lambda)$ corresponding to $\mathcal{G}_\Lambda$

cardinality affect the computation time needed to generate the automaton. Finally, we investigate the average cardinality of the dominant set as a function of the cardinality of a set of constraints.

5.1 Comparing `WeaklyHard.jl` and `WHRTgraph`

The literature contribution that is closest to our research is **`WHRTgraph`** [Linsenmayer and Allgower, 2017; Linsenmayer et al., 2021]. **`WHRTgraph`**'s analysis of weakly-hard tasks is also based on the construction of automata. While **`WHRTgraph`** handles only one weakly-hard constraint at a time, it can construct the automaton that correspond to **`AnyHit`** and **`RowHit`** constraints, making it the reference in terms of analysis capabilities. **`WHRTgraph`** is implemented in MATLAB, while **`WeaklyHard.jl`** is implemented in Julia. Hence, comparing the execution times

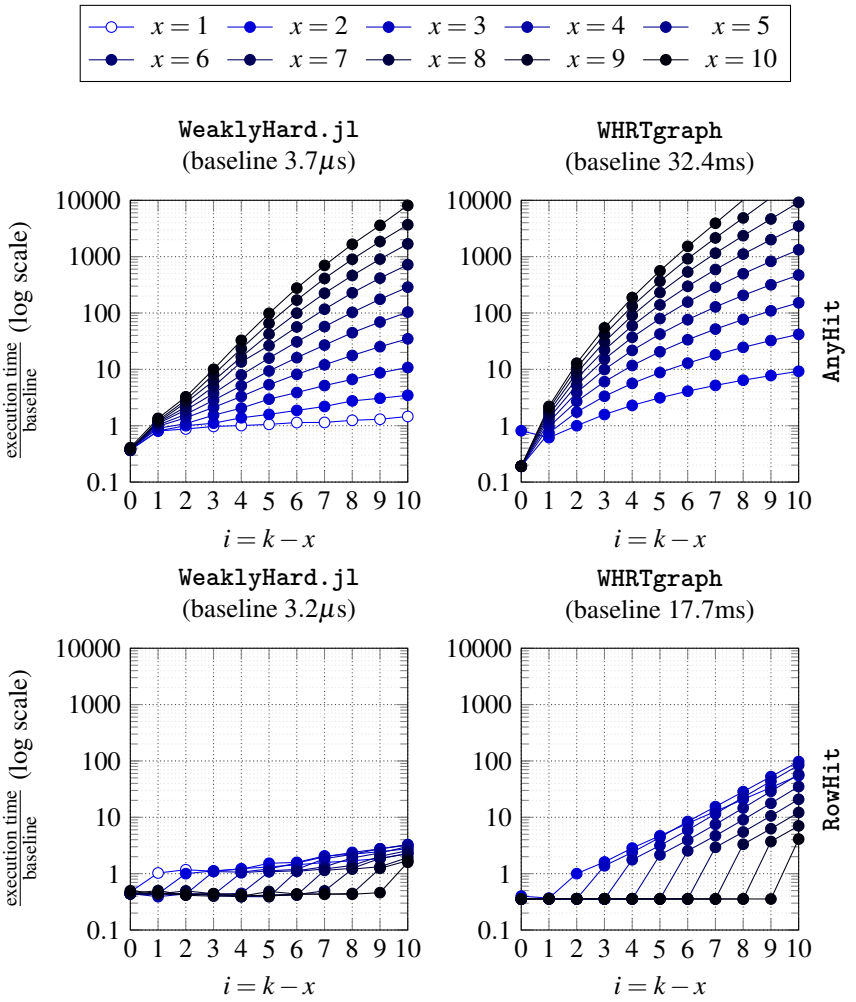


Figure 2. Execution time comparison for `AnyHit` and `RowHit` constraints with `WeaklyHard.jl` and `WHRTgraph` [Linsenmayer and Allgower, 2017] increasing the difference between window size and number of hits constrained. Baseline values are reported on top of the corresponding plots.

of the two (on their own) is pointless. Furthermore, we are more interested in assessing the scalability to an increase in the constraint window size than the absolute numbers for the execution times. We therefore define a baseline case, for a fair comparison, i.e., the reported results are fractions and multiples of the baseline, which is different for each tool and constraint type.

To test the scalability of the automaton generation, we ask both `WeaklyHard.jl` and `WHRTgraph` to generate the automata that correspond to the `AnyHit` $\binom{x}{k}$ and `RowHit` $\left\langle {x \atop k} \right\rangle$ constraints for $x \in \{1,2,\ldots,10\}$, $k = x+i$ and $i \in \{0,1,\ldots,10\}$. We divide the obtained results by the baseline value, i.e., the execution time needed for the corresponding tool to generate the automaton for the given constraint type, $x = 2$ and $k = 4$.[5]

Figure 2 shows the mean value of the execution time for the automaton generation, divided by the corresponding baseline value, using a logarithmic y-axis. The baseline computation times for `AnyHit` constraint are 3.7μs for `WeaklyHard.jl` and 32.4ms for `WHRTgraph`. On the contrary, for a `RowHit` constraint, the baseline computation time is 3.2μs for `WeaklyHard.jl` and 17.7ms for `WHRTgraph`. Due to the extensive computational time necessary to build the automata using `WHRTgraph`, each automaton was built 30 times (i.e., each point in the figure is the mean of 30 executions). `WeaklyHard.jl` is significantly faster, thus, each automata was built 100 000 times to reduce the execution time variance.

`WHRTgraph` represents a weakly-hard constraint with a slightly different, yet equivalent automaton to the one generated by `WeaklyHard.jl`. In particular, the automaton generated by `WHRTgraph` has fewer vertices and weights on the edges encode the number of consecutive deadline misses allowed between the vertices. Thus, a transition between two vertices in `WHRTgraph` is not equivalent to one outcome (as for `WeaklyHard.jl`), reducing flexibility, i.e., making it harder for example to automatically generate code to monitor the outcomes of task executions. Multiple successive outcomes for each transition also complicate the handling of sets of weakly-hard constraints. In terms of scalability, an automaton representation with fewer nodes may sound more efficient. However, we show that `WeaklyHard.jl` scales better than `WHRTgraph` by more than an order of magnitude. The baseline numbers show that `WeaklyHard.jl` is also significantly faster in absolute terms.

Comparing the scalability of the two tools for `AnyHit` constraints (leftmost plots), we observe that `WeaklyHard.jl` is more than an order of magnitude faster than `WHRTgraph`. On the contrary, for `RowHit` constraints (rightmost plots), we experience a speedup of almost two orders of magnitude for high values of $i = k - x$. The scalability of the `RowHit` constraints are further investigated in the following subsection.

5.2 Evaluating `RowHit` constraints

In the previous subsection we discussed the scalability of `WeaklyHard.jl` compared to the state-of-the-art. Despite improvements of more than an order of magnitude (not considering the baseline), the time necessary to construct the automata

[5] The choice of the baseline case reflects the simplest constraint that is correctly handled by both `WeaklyHard.jl` and `WHRTgraph`. Comparing the methods, we unveiled that `WHRTgraph` is unable to find an automaton for constraints in which $x = 1$. The two plots for `WHRTgraph` in Figure 2 do not contain results for $x = 1$ (white filled markers) precisely due to this problem.

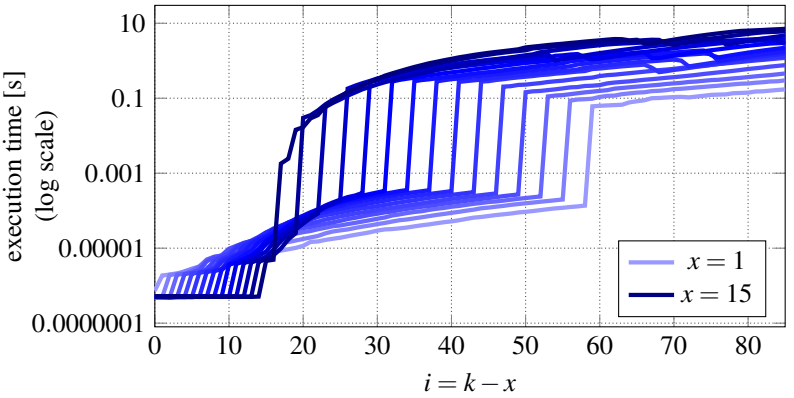


Figure 3. Mean execution time of the generation `RowHit` constraint automaton.

for `AnyHit` constraints grows rapidly with increasing window lengths. Motivated by the ongoing discussion on the practical importance of consecutive deadline hits [Åkesson et al., 2020; Vreman et al., 2021] and the scalability considerations presented in Section 4.3, we now perform an extensive evaluation of the scalability of the `RowHit` constraints.

Using `WeaklyHard.jl`, we generate the automaton corresponding to the `RowHit` $\langle {}^{x}_{k} \rangle$ constraints for $x \in \{1, 2, \ldots, 15\}$, $k \in \{x, x+1, \ldots, 100\}$. To the best of our knowledge, this is the first research work that generates automata representations of weakly-hard constraints with window lengths above 100. Figure 3 displays the mean execution time over 100 executions for the automata generation using a logarithmic scale, showing a piecewise exponential growth of execution time with some jumps. Despite having constraints with window lengths up to $k = 100$, the worst reported execution time is below 7 seconds; reinforcing the arguments in favour of using `RowHit` rather than `AnyHit` constraints.

Another interesting consideration is related to the jumps in the execution time that each line shows when reaching certain values of x and k. This follows from the choice of using integers to represent words in `WeaklyHard.jl`. For constraints where $2x + k \geq 64$, 64 bit integers are not enough to represent all sequences, and `WeaklyHard.jl` consequently converts the sequence representation to big integers (using more than 64 bits). This representation requires additional resources (memory and computation), hence producing execution time jumps.

5.3 Analysing sets of weakly-hard constraints

`WeaklyHard.jl` is the first tool that provides the ability to analyse sets of weakly-hard constraints. In the following we conduct a sensitivity analysis to assess the scalability of the automaton generation for a set of weakly hard constraints. In particular, we are interested in finding how the window size affects the execution time

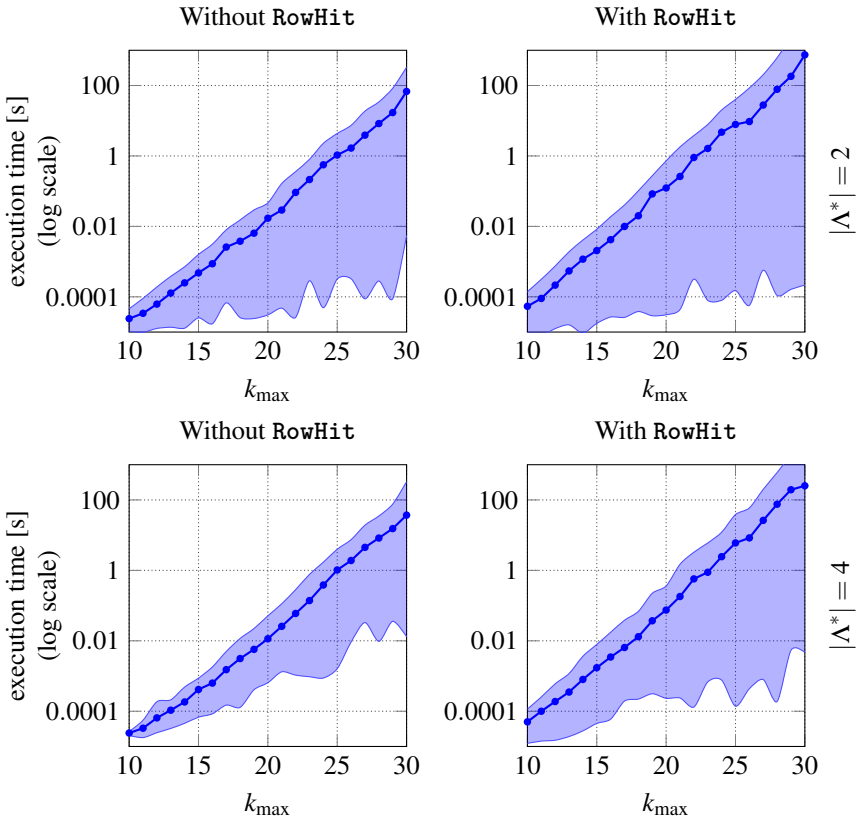


Figure 4. Execution time comparison for the generation of the automaton for sets of constraints with increasing maximum window sizes $\max k$. Average values are reported alongside the areas between minimum and maximum execution times.

of the tool, and how the composition of the set influences the execution time.

We randomise dominant sets of constraints, imposing that at least one of the constraints has a window size of $k_{\max} \in \{10, 11, \ldots, 30\}$. We generate sets with either $|\Lambda^*| = 2$ or $|\Lambda^*| = 4$. We allow these sets to include one `RowHit` constraint or none. The results of our study are shown in Figure 4. For each of the values of $k_{\max}$ in the figure, we generate 50 dominant sets Λ^*. The figure shows the average execution time in seconds (as a line) and the area representing the span between minimum and maximum execution time.

The first conclusion that we can draw is that the average execution times follow straight lines in a logarithmic scale, thus clearly pointing to the exponential time complexity inherent to expressive task models, such as the weakly-hard model [Stigge and Yi, 2015].

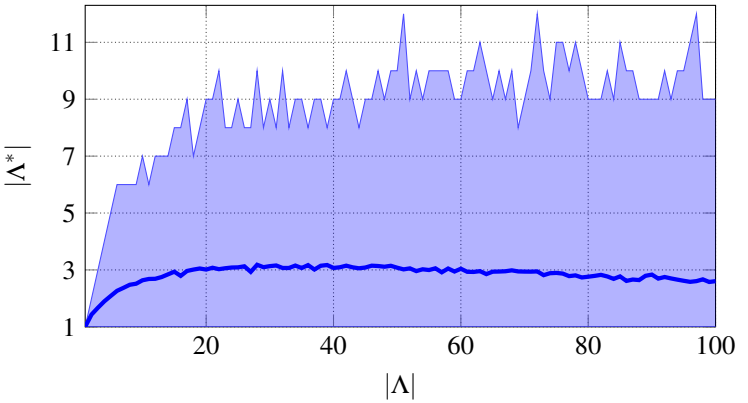


Figure 5. Average cardinality of the dominant set Λ^* as a function of $|\Lambda|$ with $k_{\max} = 100$ for 1000 randomly generated constraint sets Λ.

When the cardinality of the set $|\Lambda^*|$ increases (i.e., comparing the two leftmost and the two rightmost plots) the maximum execution time does not change significantly. In fact, states that would have been reachable with fewer constraint become unreachable due to the additional constraints pruning the state-space. However, we experience a slight reduction in the execution time's variance, which follows from the nature of the dominant set. Comparing two dominant sets, Λ_1^* and Λ_2^*, with the same $k_{\max}$: when $|\Lambda_1^*| = 2$ and $|\Lambda_2^*| = 4$, the set Λ_2^* must include less restrictive constraints (otherwise they would dominate the other constraints in the set). Hence, the set Λ_2^* is less likely to be trivial to analyse.

Finally, including a `RowHit` constraint in the set Λ^* increases the execution time by an order of magnitude. This follows from the complex interconnections between the `RowHit` and remaining weakly-hard constraints. Particularly, for the `AnyHit`, `AnyMiss`, and `RowMiss` constraints it is sufficient to count the deadline hits of the jobs currently in the window; however, the `RowHit` constraints need to keep additional track of when they appeared. This is further reinforced by the fact that when a dominant set includes a `RowHit` constraint, the other constraints in the set have to be very conservative in order to neither dominate nor be dominated by it. However, we remark that `WeaklyHard.jl` is able to generate an automaton for a set Λ^* of 4 constraints with $k_{\max} = 30$, including a `RowHit` constraint, in less than 200 seconds.

5.4 Determining the dominant constraint set

In Section 5.3 we investigated dominant sets Λ^* with cardinality $|\Lambda^*| \in \{2,4\}$. Here we justify why this is a relevant benchmark despite the low cardinality.

We select a maximum window size $k_{\max} = 100$. The window size is large enough that we can find an expressive variety of constraints without partial ordering. We

randomly generate sets Λ containing $|\Lambda| \in \{1, \ldots, 100\}$ constraints. For each value of $|\Lambda|$ we generate 1000 different sets, excluding all the trivial constraints that would reduce to $\overline{\lambda}$ and $\underline{\lambda}$. We then compute the dominant set Λ^* corresponding to each set. Figure 5 shows the average cardinality of Λ^* (solid line) and the experienced range (area).

As can be seen, most constraint sets reduce to dominant sets with cardinality less than 4, thus motivating our investigation of the automaton generation execution time. Generally, it is also interesting that additional constraints tends to reduce the cardinality of Λ^*, after a peak is reached. This is however not surprising seeing as adding constraints increases the chances of the added constraints being dominant over some of the constraints in the set.

6. Conclusion

The research behind this paper is motivated by the attention the weakly-hard model is receiving in both academic and industrial contexts. The paper primarily proposes two contributions: (i) two novel theorems that complete the relation graph between weakly-hard constraints of different types, and (ii) an open-source tool, `WeaklyHard.jl`, that helps in the analysis of weakly-hard tasks. The tool includes functions to relate different weakly-hard constraints to one another, and functions to generate automata that encode the feasible outcomes of weakly-hard tasks.

We envision `WeaklyHard.jl` to be used for (i) the analysis of complex tasksets, in which tasks are subject to different weakly-hard constraints, possibly with large windows, (ii) the generation of monitoring code that provides runtime checks for the satisfaction of weakly-hard constraints. As an example, to validate the conjectures that became the theorems of Section 3.1, we used `WeaklyHard.jl` to generate the satisfaction sets for various pairs of `AnyHit` and `RowHit` constraints. We then calculated the intersection between the generated sets to verify that our conjecture held for the specific cases under test.

We analyse the scalability of `WeaklyHard.jl` and the dominance between different constraints. Furthermore, we build dominant sets of constraints. To the best of our knowledge, `WeaklyHard.jl` is the first tool that enables the analysis of tasks that satisfy sets of weakly-hard constraints.

Acknowledgements

The authors are members of the ELLIIT Strategic Research Area at Lund University. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Number 871259 (ADMORPH project). This publication reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

References

Ahrendts, L., S. Quinton, T. Boroske, and R. Ernst (2018). "Verifying weakly-hard real-time properties of traffic streams in switched networks". In: *30th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 15:1–15:22. ISBN: 978-3-95977-075-0.

Åkesson, B., M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis (2020). "An empirical survey-based study into industry practice in real-time systems". In: *41st IEEE Real-Time Systems Symposium (RTSS)*.

Behrouzian, A., H. Ara, M. Geilen, D. Goswami, and T. Basten (2020). "Firmness analysis of real-time tasks". *ACM Trans. Embed. Comput. Syst.* **19**:4. ISSN: 1539-9087. DOI: 10.1145/3398328. URL: https://doi.org/10.1145/3398328.

Bernat, G., A. Burns, and A. Liamosi (2001). "Weakly hard real-time systems". *IEEE Transactions on Computers* **50**:4, pp. 308–321. DOI: 10.1109/12.919277.

Bernat, G. (1998). *Specification and analysis of weakly hard real-time systems*. PhD thesis. Department de les Ciéncies Matemátiques i Informática, Universitat de les Illes Balears, Spain.

Bezanson, J., A. Edelman, S. Karpinski, and V. Shah (2017). "Julia: a fresh approach to numerical computing". *SIAM review* **59**:1, pp. 65–98. URL: https://doi.org/10.1137/141000671.

Bozhko, S., G. von der Brüggen, and B. B. Brandenburg (2021). "Monte carlo response-time analysis". In: *IEEE Real-Time Systems Symposium (RTSS)*, pp. 342–355. DOI: 10.1109/RTSS52674.2021.00039.

Brüggen, G. von der, N. Piatkowski, K.-H. Chen, J.-J. Chen, and K. Morik (2018). "Efficiently approximating the probability of deadline misses in real-time systems". In: Altmeyer, S. (Ed.). *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 6:1–6:22. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.6. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8997.

Brüggen, G. von der, N. Piatkowski, K.-H. Chen, J.-J. Chen, K. Morik, and B. B. Brandenburg (2021). "Efficiently approximating the worst-case deadline failure probability under edf". In: *IEEE Real-Time Systems Symposium (RTSS)*, pp. 214–226. DOI: 10.1109/RTSS52674.2021.00029.

Burns, A. and R. Davis (2013). "Mixed criticality systems – a review". *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69.

Buttazzo, G., G. Lipari, L. Abeni, and M. Caccamo (2005). *Soft Real-Time Systems*. Springer.

Casini, D., T. BlaSS, I. Lütkebohle, and B. Brandenburg (2019). "Response-time analysis of ROS2 processing chains under reservation-based scheduling". In: *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Vol. 133, 6:1–6:23. ISBN: 978-3-95977-110-8. DOI: 10.4230/LIPIcs.ECRTS.2019.6. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10743.

Fersman, E., P. Pettersson, and W. Yi (2002). "Timed automata with asynchronous processes: schedulability and decidability". In: Katoen, J.-P. et al. (Eds.). *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 67–82. ISBN: 978-3-540-46002-2.

Fersman, E., P. Krcal, P. Pettersson, and W. Yi (2007). "Task automata: schedulability, decidability and undecidability". *Information and Computing* **205**:8, pp. 1149–1172. ISSN: 0890-5401. DOI: 10.1016/j.ic.2007.01.009.

Hamdaoui, M. and P. Ramanathan (1995). "A dynamic priority assignment technique for streams with (m,k)-firm deadlines". *IEEE Transactions on Computers* **44**:12, pp. 1443–1451.

Hammadeh, Z. A. H., R. Ernst, S. Quinton, R. Henia, and L. Rioux (2017a). "Bounding deadline misses in weakly-hard real-time systems with task dependencies". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*, pp. 584–589.

Hammadeh, Z. A. H., S. Quinton, M. Panunzio, R. Henia, L. Rioux, and R. Ernst (2017b). "Budgeting under-specified tasks for weakly-hard real-time systems". In: *29th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 76. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 17:1–17:22. ISBN: 978-3-95977-037-8.

Hammadeh, Z., R. Ernst, S. Quinton, R. Henia, and L. Rioux (2017c). "Bounding deadline misses in weakly-hard real-time systems with task dependencies". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 584–589. DOI: 10.23919/DATE.2017.7927054.

Hopcroft, J., R. Motwani, and J. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA. ISBN: 0321455363.

Horssen, E. P. van, A. R. B. Behrouzian, D. Goswami, D. Antunes, T. Basten, and W. P. M. H. Heemels (2016). "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses". In: *2016 European Control Conference (ECC)*, pp. 2571–2577. DOI: 10.1109/ECC.2016.7810677.

Huang, C., W. Li, and Q. Zhu (2019a). "Formal verification of weakly-hard systems". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19. Association for Computing Ma-

chinery, Montreal, Quebec, Canada, pp. 197–207. ISBN: 9781450362825. DOI: 10.1145/3302504.3311811.

Huang, C., K. Wardega, W. Li, and Q. Zhu (2019b). "Exploring weakly-hard paradigm for networked systems". In: *Proceedings of the Workshop on Design Automation for CPS and IoT*. DESTION '19. Association for Computing Machinery, New York, NY, USA, pp. 51–59. ISBN: 9781450366991. DOI: 10.1145/3313151.3313165.

Koren, G. and D. Shasha (1995). "Skip-Over: algorithms and complexity for overloaded systems that allow skips". In: *16th IEEE Real-Time Systems Symposium (RTSS)*, pp. 110–117.

Linsenmayer, S. and F. Allgower (2017). "Stabilization of networked control systems with weakly hard real-time dropout description". In: *56th IEEE Conference on Decision and Control (CDC)*, pp. 4765–4770.

Linsenmayer, S., B. W. Carabelli, S. Wildhagen, K. Rothermel, and F. Allgöwer (2021). "Controller and triggering mechanism co-design for control over time-slotted networks". *IEEE Transactions on Control of Network Systems* **8**, pp. 222–232.

Maggio, M., A. Hamann, E. Mayer-John, and D. Ziegenbein (2020). "Control-system stability under consecutive deadline misses constraints". In: *32nd Euromicro Conference on Real-Time Systems (ECRTS)*. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Manolache, S., P. Eles, and Z. Peng (2004). "Optimization of soft real-time systems with deadline miss ratio constraints". In: *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 562–570. DOI: 10.1109/RTTAS.2004.1317304.

Osch, M. van and S. Smolka (2001). "Finite-state analysis of the can bus protocol". In: *Proceedings Sixth IEEE International Symposium on High Assurance Systems Engineering. Special Topic: Impact of Networking*, pp. 42–52. DOI: 10.1109/HASE.2001.966806.

Pazzaglia, P., A. Hamann, D. Ziegenbein, and M. Maggio (2021a). "Adaptive design of real-time control systems subject to sporadic overruns". In: *Design, Automation & Test in Europe Conference Exhibition (DATE)*.

Pazzaglia, P., L. Pannocchi, A. Biondi, and M. D. Natale (2018). "Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses". In: Altmeyer, S. (Ed.). *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:22. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.10.

Pazzaglia, P., Y. Sun, and M. Di Natale (2021b). "Generalized weakly hard schedulability analysis for real-time periodic tasks". *ACM Trans. Embed. Comput. Syst.* **20**:1, 3:1–3:26. DOI: 10.1145/3404888. URL: https://doi.org/10.1145/3404888.

Ramanathan, P. (1999). "Overload management in real-time control applications using (m, k)-firm guarantee". *IEEE Transactions on Parallel and Distributed Systems* **10**:6, pp. 549–559. DOI: 10.1109/71.774906.

Stigge, M. and W. Yi (2015). "Graph-based models for real-time workload: a survey." *Real-Time Systems* **51**, pp. 602–636. DOI: 10.1007/s11241-015-9234-z.

Sun, Y. and M. D. Natale (2017). "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks". *ACM Transactions on Embedded Computing Systems* **16**:5s. ISSN: 1539-9087.

Tu, G., J.-l. Li, F.-m. Yang, and W. Luo (2007). "Relationships between window-based real-time constraints". In: *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, pp. 394–399. DOI: 10.1109/RTCSA.2007.62.

Vreman, N., A. Cervin, and M. Maggio (2021). "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.15.

Wu, S.-L., C.-Y. Bai, K.-C. Chang, Y.-T. Hsieh, C. Huang, C.-W. Lin, E. Kang, and Q. Zhu (2020). "Efficient system verification with multiple weakly-hard constraints for runtime monitoring". In: Deshmukh, J. et al. (Eds.). *Runtime Verification*. Springer International Publishing, Cham, pp. 497–516. ISBN: 978-3-030-60508-7.

Xu, W., Z. A. H. Hammadeh, A. Kröller, R. Ernst, and S. Quinton (2015). "Improved deadline miss models for real-time systems using typical worst-case analysis". In: *27th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 247–256.

Zeng, H. and M. Di Natale (2012). "Schedulability analysis of periodic tasks implementing synchronous finite state machines". In: *24th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 353–362. DOI: 10.1109/ECRTS.2012.30.

Paper IV

Stability of Linear Control Systems under Extended Weakly-Hard Constraints

Nils Vreman Paolo Pazzaglia Victor Magron Jie Wang
Martina Maggio

Abstract

Control systems can show robustness to many events, like disturbances and model inaccuracies. It is natural to speculate that they are also robust to sporadic deadline misses when implemented as digital tasks on an embedded platform. This paper proposes a comprehensive stability analysis for control systems subject to deadline misses, leveraging a new formulation to describe the patterns experienced by the control task under different handling strategies. Such analysis brings the assessment of control systems robustness to computational problems one step closer to the controller implementation.

Originally published in IEEE Control Systems Letters (2022). The mathematical notation has been unified to match the remainder of the thesis. Reprinted with permission.

1. Introduction

Robustness is an essential concern in the design of control systems; they must be able to reliably handle nonlinear effects, unmodeled dynamics and noise, as well as delays in signal transmissions and dropped packets. A lesser known problem concerns the assessment of robustness to *computational issues* when controllers are implemented as periodic tasks in cheap embedded platforms. Such tasks are expected to execute with real-time guarantees, i.e., their execution must be completed before a well-defined *deadline*, when the control output must be sent to the actuator. However, it is common in practice [Akesson et al., 2020] that tasks do not always complete within their deadline, causing what is called a *deadline miss*. This may be caused by delays in computation and memory accesses, transient overloads, bugs and other issues.

A popular model to describe real-time systems allowing deadline misses is the *weakly-hard* model [Bernat et al., 2001]. Weakly-hard tasks feature constraints defining a maximum number of deadlines that can be missed (alternatively, a minimum number to be satisfied) in a given number of consecutive periods. This model is also the focus of this work. To analyse the effects on the controlled plant, it is necessary to specify also *what happens when the miss is experienced*, both in terms of changes to the control signal and of actions taken to deal with the failed computation [Pazzaglia et al., 2019]. An instance that experiences a deadline miss can be allowed to continue executing until completion (and possibly used later), while in other applications it is stopped and discarded instead.

There is however a mismatch between the guarantees that can be obtained for real-time tasks and platforms [Xu et al., 2015; Choi et al., 2019], and the analysis available for *control* tasks under the weakly-hard model. Fewer works deal with *stability* analysis of weakly-hard real-time control tasks, often targeting specific use-cases. For instance, the analysis in [Maggio et al., 2020] is limited to constraints specifying a maximum number of *consecutive* deadline misses. The results in [Linsenmayer and Allgower, 2017; Linsenmayer et al., 2020], obtained for networked linear control systems having packet dropouts bounded using the weakly-hard model, can not be generalised for *late completions* or *sets* of weakly-hard constraints. The authors of [Liang et al., 2019; Liang et al., 2020] studied safety guarantees of weakly-hard controllers, considering a miss as a discarded computation with a known periodic pattern. In [Huang et al., 2020; Huang et al., 2019], an over-approximation-based approach is proposed to check the safety of nonlinear weakly-hard systems, where misses are treated as discarded computations and the actuator holds its previous value. Convergence rates (providing sufficient stability guarantees) are analysed in [Gaukler et al., 2019]. A Lyapunov-based stability analysis of nonlinear weakly-hard systems is studied in [Hertneck et al., 2021], with deadline misses treated as packet dropouts. However, the state-of-the-art listed above lack generalisability to more expressive real-time implementations, such as different deadline miss models or handling strategies.

This paper aims at filling the gap, by providing a stability analysis that can be applied to a class of generic weakly-hard models and deadline miss handling strategies. First, we formally extend the weakly-hard model to explicitly consider the strategy used to handle the miss events. By leveraging an automaton representation of the sequences allowed by (a set of) extended weakly-hard constraints, we use Kronecker lifting and the joint spectral radius to properly express its stability conditions. Using the concept of constraint dominance, we prove analytic bounds on the stability of a weakly-hard system with respect to *less dominant* constraints. Finally, we analyse the stability of the resulting closed-loop systems using `SparseJSR` [Wang et al., 2021a], which exploits the sparsity pattern that naturally arises in the Kronecker lifted representation. The proposed analysis calls for modularity and separation of concern, and can be a useful tool to decouple the constraint specification and the control verification.

2. Background and Notation

We consider a controllable and fully observable *discrete-time* sampled linear time invariant system, expressed as

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

where $x_k \in \mathbb{R}^{n_x}$, $u_k \in \mathbb{R}^{n_u}$ and $y_k \in \mathbb{R}^{n_y}$ are the plant state, the control signal and the plant output, sampled at time $k \cdot T$, T is the sampling period, and $k \in \mathbb{N}_{\geq}$. The plant is controlled by a stabilising, LTI, one-step delay discrete-time controller

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

where $z_k \in \mathbb{R}^{n_z}$ is the controller's internal state and $r_k \in \mathbb{R}^{n_y}$ is the setpoint. Without loss of generality, we consider $r_k = 0$.

2.1 Real-time tasks that may miss deadlines

The controller in (2) is implemented as a real-time task τ, and designed to be executed periodically with period T in a real-time embedded platform. Under nominal conditions the task releases an instance (called *job*) in each period, that should be completed before the release of the next instance. We denote the sequence of activation instants for τ with $(a_k)_{k\in\mathbb{N}_{\geq}}$, such that, in nominal conditions, $a_{k+1} = a_k + T$, the sequence of completion instants $(f_k)_{k\in\mathbb{N}_{\geq}}$, and the sequence of job deadlines with $(d_k)_{k\in\mathbb{N}_{\geq}}$, such that $d_k = a_k + T$ (also called *implicit* deadline). This requirement can be either satisfied or not, leading respectively to deadline hits and misses.

DEFINITION 1—DEADLINE HIT AND MISS
The k-th job of a periodic task τ with period T hits its deadline when $f_k \leq d_k$ and misses its deadline when $f_k > d_k$.

We refer to both deadline hits and misses using the term *outcome* of a job. Intuitively, each job's outcome is dependent on the characteristics of the remaining tasks executing in the real-time system and the chosen scheduling algorithm. Given a taskset and a (worst-case) schedule, it is possible to bound the worst-case behaviour of the job outcomes [Bernat et al., 2001; Xu et al., 2015]. This bound is generally denoted using the *weakly-hard model* [Bernat et al., 2001]. Following such model, a task τ may satisfy any combination of these weakly-hard constraints, defined as follows.

(i) $\tau \vdash \overline{\binom{x}{\ell}}$: in any window of ℓ consecutive jobs, at most x deadlines are missed;

(ii) $\tau \vdash \binom{x}{\ell}$: in any window of ℓ consecutive jobs, at least x deadlines are hit;

(iii) $\tau \vdash \overline{\langle {}^{x}_{\ell} \rangle}$: in any window of ℓ consecutive jobs, at most x *consecutive* deadlines are missed; and

(iv) $\tau \vdash \langle {}^{x}_{\ell} \rangle$: in any window of ℓ consecutive jobs, at least x *consecutive* deadlines are hit.

In all such cases, $x \in \mathbb{N}_{\geq}$, $\ell \in \mathbb{N}_{>}$, and $x \leq \ell$. A generic weakly-hard constraint is hereafter denoted with the symbol λ, while a set of L constraints will be referred to as $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_L\}$.

We define a *word* $w = \langle c_1, c_2, \ldots, c_N \rangle$ as a sequence of N consecutive outcomes, where each outcome c_k is a character in the alphabet $\Sigma = \{\mathtt{M}, \mathtt{H}\}$. We use $w \vdash \lambda$ to denote that w satisfies the constraint λ. Stating that $\tau \vdash \lambda$ means that all the possible sequences of outcomes that τ can experience satisfy the corresponding constraint λ. The set of such sequences naturally results from the definition of λ, and is formally defined as the *satisfaction set* as follows [Bernat et al., 2001].

DEFINITION 2—SATISFACTION SET $\mathcal{S}_N(\lambda)$
We denote with $\mathcal{S}_N(\lambda)$ the set of words of length $N \geq 1$ that satisfy a constraint λ. Formally, $\mathcal{S}_N(\lambda) = \{w \in \Sigma^N \mid w \vdash \lambda\}$.

Taking the limit to infinity, the set $\mathcal{S}(\lambda)$ contains all the words of infinite length that satisfy λ. The notion of *domination* between constraints [Bernat et al., 2001] then follows.

DEFINITION 3—CONSTRAINT DOMINATION
Constraint λ_i dominates λ_j *(formally, $\lambda_i \preceq \lambda_j$) if $\mathcal{S}(\lambda_i) \subseteq \mathcal{S}(\lambda_j)$.*

2.2 Control tasks that may miss deadlines

When a control task τ is implemented on an embedded platform with limited computational power, alongside other applications, it is not uncommon for it to experience deadline misses, even in case of simple control designs (PID, LQG, etc) [Akesson et al., 2020; Pazzaglia et al., 2021]. Computational overruns may be caused by, e.g., bursts of interrupts, cache misses, variable execution times of ancillary functions, or other complex interactions. If such events are rare or temporary, choosing a longer period for the controller to avoid them may result in worse performance and stability margins for nominal conditions [Pazzaglia et al., 2019].

Characterising the stability and performance of such controllers requires knowing what happens when a control deadline is missed [Pazzaglia et al., 2019; Maggio et al., 2020; Vreman et al., 2021]. In particular, we need a *deadline miss handling strategy* to decide the fate of the job that missed the deadline (and possibly the next ones), and an *actuator mode* to deal with the loss of a new control signal, for example by `Hold`ing the previous value constant or `Zero`ing it [Schenato, 2009]. A few handling strategies for periodic controllers have been proposed in literature, the most interesting being `Kill` and `Skip` [Cervin, 2005; Pazzaglia et al., 2019; Maggio et al., 2020].

DEFINITION 4—`Kill` STRATEGY
Under the `Kill` *strategy, a job that misses its deadline is terminated immediately. Formally, for the k-th job of* τ *either* $f_k \leq d_k$ *or* $f_k = \infty$.

DEFINITION 5—`Skip` STRATEGY
Under the `Skip` *strategy, a job that misses its deadline is allowed to continue during the following period. Formally, if the k-th job of* τ *misses its deadline* d_k*, a new deadline* $d_k^+ = d_k + T$ *is set for the job, and* $a_{k+1} = d_k^+$.

2.3 Stability analysis techniques based on JSR

In [Maggio et al., 2020], the authors identify a set of subsequences of hit and missed deadlines, which can be arbitrarily combined to obtain all possible sequences in $\mathcal{S}(\overline{\langle {}^{x}_{\ell} \rangle})$. The stability analysis of the resulting arbitrary switching system is then obtained by leveraging the *Joint Spectral Radius* (JSR) [Rota and Strang, 1960].

Given $m \in \mathbb{N}_{>}$ and a set of matrices $\mathcal{A} = \{\Phi_1, \ldots, \Phi_m\} \subseteq \mathbb{R}^{n \times n}$, under the hypothesis of arbitrary switching over any sequence $s = \langle a_1, a_2, \ldots \rangle$ of indices of matrices in $\mathcal{A}$, the JSR of $\mathcal{A}$ is defined by:

$$\rho(\mathcal{A}) = \lim_{N \to \infty} \max_{s \in \{1,\ldots,m\}^N} \|\Phi_{a_N} \cdots \Phi_{a_2} \Phi_{a_1}\|^{\frac{1}{N}}. \tag{3}$$

The number $\rho(\mathcal{A})$ characterizes the maximal asymptotic growth rate of matrix products from $\mathcal{A}$ (thus $\rho(\mathcal{A}) < 1$ means that the system is asymptotically stable), and is independent of the norm $\|\cdot\|$ used in (3). Existing practical tools such as the

JSR Matlab toolbox [Vankeerberghen et al., 2014] include multiple algorithms to compute both upper and lower bounds on $\rho(\mathcal{A})$.

When the switching sequences between the dynamics of $\mathcal{A}$ are not arbitrary, but constrained by a graph $\mathcal{G}$, the so called *constrained joint spectral radius* (CJSR) [Dai, 2012] can be applied. Introducing $S_N(\mathcal{G})$ as the set of all possible switching sequences s of length N that satisfy the constraints of a graph $\mathcal{G}$, the CJSR of $\mathcal{A}$ is defined by

$$\rho(\mathcal{A},\mathcal{G}) = \lim_{N\to\infty} \max_{s\in S_N(\mathcal{G})} \|\Phi_{a_N}\cdots\Phi_{a_2}\Phi_{a_1}\|^{\frac{1}{N}}. \tag{4}$$

In general, computing or approximating the CJSR is harder than using the JSR. In [Philippe et al., 2016], the authors propose a multinorm-based method to approximate with arbitrary accuracy the CJSR. Other works [Kozyakin, 2014; Xu and Acikmese, 2020] 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 will be also our approach, as detailed later.

In [Parrilo and Jadbabaie, 2008], the authors propose an efficient approach to compute upper bounds of the JSR based on positive polynomials which can be decomposed as *sums of squares* (SOS). Finding the coefficients of a polynomial being SOS simplifies to solving an SDP [Lasserre, 2001]. To reduce time and space complexity, a *sparse* variant has been proposed in [Wang et al., 2021a] exploiting the sparsity of the input matrices, based on the *term sparsity* SOS (TSSOS) framework [Wang et al., 2021b]. By contrast, the procedure in [Parrilo and Jadbabaie, 2008] will be denoted hereafter as *dense*. While providing a more conservative result, the sparse upper bound can be obtained significantly faster if the matrices from $\mathcal{A}$ are sparse [Wang et al., 2021a], e.g., the matrices we analyse in Section 5.

3. Extended Weakly-Hard Task Model

To provide a comprehensive analysis framework, we need to examine what occurs in each time interval $(\pi_k)_{k\in\mathbb{N}_\geq}$, with $\pi_k = [a_0 + k\cdot T, a_0 + (k+1)\cdot T)$. In this context, an extension of the weakly-hard model is required to account for the given deadline miss handling strategy, denoted with the symbol $\mathcal{H}$.

DEFINITION 6—EXTENDED WEAKLY-HARD MODEL $\tau \vdash \lambda^{\mathcal{H}}$
A task τ may satisfy any combination of the four extended weakly-hard constraints *(EWHC) $\lambda^{\mathcal{H}}$:*

(i) $\tau \vdash \overline{\binom{x}{\ell}}^{\mathcal{H}}$*: in any window of ℓ consecutive jobs, at most x intervals lack a job completion;*

(ii) $\tau \vdash \binom{x}{\ell}^{\mathcal{H}}$*: in any window of ℓ consecutive jobs, at least x intervals have a job completion;*

(iii) $\tau \vdash \overline{\langle {}^{x}\backslash_{\ell} \rangle}^{\mathcal{H}}$*: in any window of* ℓ *consecutive jobs, at most* x consecutive *intervals lack a job completion;*

(iv) $\tau \vdash \langle {}^{x}\backslash_{\ell} \rangle^{\mathcal{H}}$*: in any window of* ℓ *consecutive jobs, at least* x consecutive *intervals have a job completion*

with $x \in \mathbb{N}_{\geq}$, $\ell \in \mathbb{N}_{>}$, *and* $x \leq \ell$, *while using strategy* $\mathcal{H}$ *to handle potential deadline misses.*

The definition above differs from the original weakly-hard model of [Bernat et al., 2001], since (i) it explicitly introduces the handling strategy $\mathcal{H}$; and (ii) it focuses on the presence of a new control command at the end of each time interval π_k, instead of checking the deadline miss events, which guarantees its applicability also for strategies different than `Kill`.

We now require an expressive alphabet $\Sigma(\mathcal{H})$ to characterize the behaviour of task τ in each possible time interval. For both `Kill` and `Skip` strategies, each interval π_k contains at most one activated and one completed job. This restricts the possible behaviours to three cases:

(i) a time interval in which the same job is both released and completed is denoted by `H` (*hit*);

(ii) a time interval in which no job is completed is denoted by `M` (*miss*);

(iii) a time interval in which no job is released, but a job (released in a previous interval) is completed, is denoted by `R` (*recovery*).

By checking all unique combinations of job activations and completions in each interval, we obtain the alphabets for `Kill` and `Skip` as $\Sigma(\texttt{Kill}) = \{\texttt{M},\texttt{H}\}$ and $\Sigma(\texttt{Skip}) = \{\texttt{M},\texttt{H},\texttt{R}\}$, respectively. The recovery character `R` is used in the `Skip` alphabet to identify the late *completion* of a job. As a consequence, `R` is treated equivalently to `H` when verifying the extended weakly hard constraints (EWHC).

The algebra presented in Section 2.1 is extended to the new alphabet. We assign a character of the alphabet $\Sigma(\mathcal{H})$ to each interval π_k. A word $w = \langle c_1, c_2, \ldots, c_N \rangle$ is used to represent a sequence of N outcomes for task τ, with $c_k \in \Sigma(\mathcal{H})$ representing the outcome associated to the interval π_k. To enforce only feasible sequences, we introduce an order constraint for the `R` character with the following Rule.

RULE 1—OUTCOME ORDERING

For any word $w \in \Sigma(\texttt{Skip})^N$, `R` *may only directly follow* `M`*, or be the initial element of the word.*

The extended weakly-hard model also inherits all the properties of the original weakly-hard model. In particular, the satisfaction set of $\lambda^{\mathcal{H}}$ can be defined for $N \geq 1$ as $\mathcal{S}_N(\lambda^{\mathcal{H}}) = \{w \in \Sigma(\mathcal{H})^N \mid w \vdash \lambda^{\mathcal{H}}\}$, and the constraint domination still holds as $\lambda_i^{\mathcal{H}} \preceq \lambda_j^{\mathcal{H}}$ if $\mathcal{S}(\lambda_i^{\mathcal{H}}) \subseteq \mathcal{S}(\lambda_j^{\mathcal{H}})$.

4. Automaton Representation of EWHC

Any EWHC, as presented in Definition 6, can be systematically represented using an *automaton*. In this paper we build upon the `WeaklyHard.jl` automaton model presented in [Vreman et al., 2022]. Here, a (minimal) automaton $\mathcal{G}_{\lambda^{\mathcal{H}}} = (V_{\lambda^{\mathcal{H}}}, E_{\lambda^{\mathcal{H}}})$ associated to $\lambda^{\mathcal{H}}$ consists of a set of vertices ($V_{\lambda^{\mathcal{H}}}$) and a set of directed labeled edges ($E_{\lambda^{\mathcal{H}}}$). Each vertex $v_i \in V_{\lambda^{\mathcal{H}}}$ corresponds to a word of outcomes of the extended weakly-hard task executions. Trivially, there exists no vertices for words that do not satisfy the EWHC. A directed labeled edge $e_{i,j} = (v_i, v_j, c) \in E_{\lambda^{\mathcal{H}}}$ (also denoted *transition*) connects two vertices iff the outcome $c \in \Sigma(\mathcal{H})$ – the edge's label – appended to the tail vertex's word representation (v_i) would result in the word equivalent to the one of the head vertex (v_j). Thus, a random walk in the automaton corresponds to a random word satisfying the EWHC. In particular, all the walks in the automaton corresponds to *all* words in $\mathcal{S}(\lambda^{\mathcal{H}})$.

Since the `WeaklyHard.jl` automaton model only uses the binary alphabet $\Sigma = \{\mathtt{M}, \mathtt{H}\}$, we require the additional character `R` to handle the `Skip` strategy properly. Recall that both a hit (`H`) and a recovery (`R`) are considered job completions. Thus, for the `Skip` strategy, we post-process the automaton by enforcing that Rule 1 is honoured and that the corresponding transitions are correct, i.e., switching the labels on some edges from `H` to `R`. We emphasise that despite the extended automaton model appear similar for the `Kill` and `Skip` strategies, the differing transitions of the two automata significantly affect the corresponding closed-loop systems, as will be clear in Section 5.

The `WeaklyHard.jl` automaton model also allows for the case where the task τ is subject to a set of multiple constraints. Since the stability analysis presented in this paper is invariant to the type (and amount) of the constraints acting on the control task τ, we henceforth say that τ is subject to a set of EWHC $\Lambda^{\mathcal{H}}$ (unless stated otherwise).

Extracting all transitions in $E_{\Lambda^{\mathcal{H}}}$ corresponding to a character $c \in \Sigma(\mathcal{H})$ yields what is generally known as a *directed adjacency matrix* [Xu and Hong, 2012], denoted here as a *transition matrix*.

DEFINITION 7—TRANSITION MATRIX
Given an automaton $\mathcal{G}_{\Lambda^{\mathcal{H}}}$, the transition matrix $F_c(\mathcal{G}_{\Lambda^{\mathcal{H}}}) \in \mathbb{R}^{n_V \times n_V}$, *with $n_V = |V_{\Lambda^{\mathcal{H}}}|$ and $c \in \Sigma(\mathcal{H})$, is computed as $F_c(\mathcal{G}_{\Lambda^{\mathcal{H}}}) = \{f_{i,j}(c)\}$ with*

$$f_{i,j}(c) = \begin{cases} 1, & \text{if } \exists e_{i,j} = (v_i, v_j, c) \in E_{\Lambda^{\mathcal{H}}} \\ 0, & \text{otherwise.} \end{cases}$$

Since *at most one* successor exists from each vertex with a transition labeled with $c \in \Sigma(\mathcal{H})$, matrix F_c will have a column sum of either 1 or 0. We now introduce a vector $q_k \in \mathbb{R}^{n_V}$ called *G-state*, with $n_V = |V_{\Lambda^{\mathcal{H}}}|$, representing the state of the given automaton $\mathcal{G}_{\Lambda^{\mathcal{H}}}$ at interval π_k.

DEFINITION 8—G-STATE q_k
Given an automaton $\mathcal{G}_{\Lambda^{\mathcal{H}}}$ and a word $w \in \Sigma(\mathcal{H})^N$, $w = \langle c_1, c_2, \ldots, c_N \rangle$, for $\ell = |v|$, $v \in V_{\Lambda^{\mathcal{H}}}$, we define $q_k \in \mathbb{R}^{n_V}$, where the i-th element $q_{k,i}$ is:

$$q_{k,i} = \begin{cases} 1, & \text{if } \langle c_{k-\ell}, \ldots, c_{k-1} \rangle \equiv v_i \in V_{\Lambda^{\mathcal{H}}} \\ 0, & \text{otherwise.} \end{cases}$$

The G-state q_k is the vector representation of the vertex *left* at step k: here, $q_k = 0$ means that the transition at step $k-1$ was infeasible for the automaton. Given an arbitrary word $w = \langle c_1, \ldots, c_k, \ldots \rangle$, the G-state dynamics is defined as $q_{k+1} = F_c(\mathcal{G}_{\Lambda^{\mathcal{H}}}) \cdot q_k$, and the following property holds [Xu and Hong, 2012].

LEMMA 1—INFEASIBLE SEQUENCE
If $w \notin \mathcal{S}_N(\Lambda^{\mathcal{H}})$, then $F_w(\mathcal{G}_{\Lambda^{\mathcal{H}}}) = F_{c_N}(\mathcal{G}_{\Lambda^{\mathcal{H}}}) \cdots F_{c_2}(\mathcal{G}_{\Lambda^{\mathcal{H}}}) \cdot F_{c_1}(\mathcal{G}_{\Lambda^{\mathcal{H}}}) = 0$

Thus, if $q_k = 0$ for an arbitrary k, then $q_{k'} = 0$ for $k' \geq k$.

5. Stability Analysis

Using the alphabet $\Sigma(\mathcal{H})$ and the chosen actuator mode (i.e., `Zero`ing, or `Hold`ing the previous value), we compute the closed-loop behaviour of the controlled system. We identify one matrix for each dynamics corresponding to an interval π_k associated by $c \in \Sigma(\mathcal{H})$, building the set $\mathcal{A}^{\mathcal{H}}$.

`Kill`: Defining $\tilde{x}_k^{\mathtt{K}} = [x_k^{\mathrm{T}} \; z_k^{\mathrm{T}} \; u_k^{\mathrm{T}}]^{\mathrm{T}}$ as the closed-loop state vector, we compute the discrete time closed-loop system dynamics $\Phi_{\mathtt{H}}^{\mathtt{K}}$, corresponding to the character `H`:

$$\tilde{x}_{k+1}^{\mathtt{K}} = \Phi_{\mathtt{H}}^{\mathtt{K}} \tilde{x}_k^{\mathtt{K}}, \quad \Phi_{\mathtt{H}}^{\mathtt{K}} = \begin{bmatrix} A & 0 & B \\ -GC & F & -GD \\ -KC & H & -KD \end{bmatrix}.$$

For the case of `M`, the controller execution terminates prematurely and its states are not updated ($z_{k+1} = z_k$). Therefore, depending on the actuation mode (`Zero` or `Hold`), the controller output is either zeroed ($u_{k+1} = 0$) or held ($u_{k+1} = u_k$). The resulting closed-loop system in state-space form is denoted with $\Phi_{\mathtt{M}}^{\mathtt{K}}$:

$$\tilde{x}_{k+1}^{\mathtt{K}} = \Phi_{\mathtt{M}}^{\mathtt{K}} \tilde{x}_k^{\mathtt{K}}, \quad \Phi_{\mathtt{M}}^{\mathtt{K}} = \begin{bmatrix} A & 0 & B \\ 0 & I & 0 \\ 0 & 0 & \Delta \end{bmatrix}.$$

Here, $\Delta = I$ (identity matrix) if the control signal is held and $\Delta = 0$ if zeroed. The set of dynamic matrices under the `Kill` strategy is then $\mathcal{A}^{\mathtt{K}} = \{\Phi_{\mathtt{H}}^{\mathtt{K}}, \Phi_{\mathtt{M}}^{\mathtt{K}}\}$.

`Skip`: For the `Skip` strategy, we introduce two additional states $\hat{x}_k$ and $\hat{u}_k$ storing the old values of x_k and u_k while the controller awaits an update. The resulting state

vector then becomes $\tilde{x}_k^{\mathtt{S}} = [x_k^{\mathrm{T}}\, z_k^{\mathrm{T}}\, u_k^{\mathrm{T}}\, \hat{x}_k^{\mathrm{T}}\, \hat{u}_k^{\mathrm{T}}]^{\mathrm{T}}$. When π_k is associated to $\mathtt{H}$, the two additional states mirror the behaviour of the states of which they are storing data. The resulting closed-loop system is described using $\Phi_{\mathtt{H}}^{\mathtt{S}}$:

$$\tilde{x}_{k+1}^{\mathtt{S}} = \Phi_{\mathtt{H}}^{\mathtt{S}} \tilde{x}_k^{\mathtt{S}}, \quad \Phi_{\mathtt{H}}^{\mathtt{S}} = \begin{bmatrix} A & 0 & B & 0 & 0 \\ -GC & F & -GD & 0 & 0 \\ -KC & H & -KD & 0 & 0 \\ A & 0 & B & 0 & 0 \\ -KC & H & -KD & 0 & 0 \end{bmatrix}.$$

For the case of $\mathtt{M}$ in π_k, $\hat{x}_k$ and $\hat{u}_k$ maintain their previous values. The resulting closed-loop is described by $\Phi_{\mathtt{M}}^{\mathtt{S}}$:

$$\tilde{x}_{k+1}^{\mathtt{S}} = \Phi_{\mathtt{M}}^{\mathtt{S}} \tilde{x}_k^{\mathtt{S}}, \quad \Phi_{\mathtt{M}}^{\mathtt{S}} = \begin{bmatrix} A & 0 & B & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & \Delta & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}.$$

Finally, for the case of $\mathtt{R}$, the new control command is calculated using the values stored in $\hat{x}_k$ and $\hat{u}_k$. The resulting closed-loop system is described by $\Phi_{\mathtt{R}}^{\mathtt{S}}$:

$$\tilde{x}_{k+1}^{\mathtt{S}} = \Phi_{\mathtt{R}}^{\mathtt{S}} \tilde{x}_k^{\mathtt{S}}, \quad \Phi_{\mathtt{R}}^{\mathtt{S}} = \begin{bmatrix} A & 0 & B & 0 & 0 \\ 0 & F & 0 & -GC & -GD \\ 0 & H & 0 & -KC & -KD \\ A & 0 & B & 0 & 0 \\ 0 & H & 0 & -KC & -KD \end{bmatrix}.$$

The resulting set of matrices under the `Skip` strategy is then defined as $\mathcal{A}^{\mathtt{S}} = \{\Phi_{\mathtt{H}}^{\mathtt{S}}, \Phi_{\mathtt{M}}^{\mathtt{S}}, \Phi_{\mathtt{R}}^{\mathtt{S}}\}$.

5.1 Kronecker lifted switching system

Combining the set of system dynamics $\mathcal{A}^{\mathcal{H}}$ with the associated automaton $\mathcal{G}_{\Lambda^{\mathcal{H}}}$, we seek to obtain an equivalent system model based on Kronecker lifting, characterized by a set of matrices denoted by $\mathcal{L}_{\Lambda^{\mathcal{H}}}$ and behaving as an *arbitrary switching system*, such that $\rho\left(\mathcal{L}_{\Lambda^{\mathcal{H}}}\right) = \rho\left(\mathcal{A}^{\mathcal{H}}, \mathcal{G}_{\Lambda^{\mathcal{H}}}\right)$. In this way, powerful algorithms applicable to arbitrary switching system [Vankeerberghen et al., 2014; Wang et al., 2021a] can be used to find tight stability bounds. We build upon the Kronecker lifting approach of [Xu and Acikmese, 2020]. Leveraging the vector q_k of Definition 8, we introduce the *lifted discrete-time state* $\xi_k \in \mathbb{R}^{n \cdot n_V}$, defined as $\xi_k = q_k \otimes \tilde{x}_k$, where $n_V = |V_{\Lambda^{\mathcal{H}}}|$ and $\otimes$ is the Kronecker product. By construction, ξ_k is a vector composed of n_V blocks of size n, where at most one block is equal to $\tilde{x}_k$ and all other blocks are equal to the 0 vector. Then, we build a set of lifted matrices $L_c(\mathcal{G}_{\Lambda^{\mathcal{H}}}) \in \mathbb{R}^{n \cdot n_V \times n \cdot n_V}$,

which incorporates both the system dynamics and the possible transitions given a certain outcome $c \in \Sigma(\mathcal{H})$:

$$L_c(\mathcal{G}_{\Lambda\mathcal{H}}) = F_c(\mathcal{G}_{\Lambda\mathcal{H}}) \otimes \Phi_c^{\mathcal{H}}, \quad c \in \Sigma(\mathcal{H}). \tag{5}$$

The lifted dynamics of the closed loop system then become $\xi_{k+1} = L_c(\mathcal{G}_{\Lambda\mathcal{H}}) \cdot \xi_k$. Formally, we obtain a system composed of a set of switching dynamic matrices, $\mathcal{L}_{\Lambda\mathcal{H}}$.

DEFINITION 9—LIFTED SWITCHING SET $\mathcal{L}_{\Lambda\mathcal{H}}$
Given a set of dynamic matrices $\mathcal{A}^{\mathcal{H}}$ and an automaton $\mathcal{G}_{\Lambda\mathcal{H}}$, the switching set $\mathcal{L}_{\Lambda\mathcal{H}}$ is defined as:

$$\mathcal{L}_{\Lambda\mathcal{H}} = \{L_c(\mathcal{G}_{\Lambda\mathcal{H}}) \mid c \in \Sigma(\mathcal{H})\}.$$

Leveraging the mixed-product property of $\otimes$ and introducing a proper submultiplicative norm, it is possible to prove that $\rho(\mathcal{L}_{\Lambda\mathcal{H}}) = \rho(\mathcal{A}^{\mathcal{H}}, \mathcal{G}_{\Lambda\mathcal{H}})$. For more details and a formal proof we refer the interested reader to [Xu and Acikmese, 2020].

5.2 Extended weakly hard and JSR properties

We now provide a general relation between *all* EWHCs in terms of the joint spectral radii.

THEOREM 1—JSR DOMINANCE
Given $\lambda_1^{\mathcal{H}}$ and $\lambda_2^{\mathcal{H}}$ as arbitrary EWHCs, if $\lambda_2^{\mathcal{H}} \preceq \lambda_1^{\mathcal{H}}$ then

$$\rho(\mathcal{L}_{\lambda_2^{\mathcal{H}}}) \leq \rho(\mathcal{L}_{\lambda_1^{\mathcal{H}}}).$$

Proof. From Equation (3), for a generic EWHC $\lambda^{\mathcal{H}}$,

$$\rho(\mathcal{L}_{\lambda\mathcal{H}}) = \lim_{N\to\infty} \rho_N(\mathcal{L}_{\lambda\mathcal{H}}), \quad \rho_N(\mathcal{L}_{\lambda\mathcal{H}}) = \max_{a\in\mathcal{S}_N(\lambda^{\mathcal{H}})} \|\Phi_a\|^{1/N}.$$

Definition 3 gave us that $\lambda_2^{\mathcal{H}} \preceq \lambda_1^{\mathcal{H}}$ iff $\mathcal{S}(\lambda_2^{\mathcal{H}}) \subseteq \mathcal{S}(\lambda_1^{\mathcal{H}})$. Thus, if for a word b it holds that $b \in \mathcal{S}_N(\lambda_2^{\mathcal{H}})$, then it also holds that $b \in \mathcal{S}_N(\lambda_1^{\mathcal{H}})$. The set of all possible Φ_b is thus included in the set of all possible Φ_a, $a \in \mathcal{S}_N(\lambda_1^{\mathcal{H}})$, thus:

$$\max_{b\in\mathcal{S}_N(\lambda_2^{\mathcal{H}})} \|\Phi_b\|^{1/N} \leq \max_{a\in\mathcal{S}_N(\lambda_1^{\mathcal{H}})} \|\Phi_a\|^{1/N}, \quad \forall N \in \mathbb{N}_{>}.$$

The theorem follows immediately when $N \to \infty$. □

Theorem 1 is the first result that provides an analytic, correlation between the control theoretical analysis and real-time implementation. Primarily, it implies that the constraint dominance from Definition 3 also carries on to the JSR, giving us a notion of *JSR dominance*. The results of Theorem 1 are strategy-independent, further

reducing the coupling between the control analysis and real-time implementation, and are also independent of the controlled system's dynamics.

Two Corollaries of Theorem 1 are derived for the commonly used models $\overline{\langle {}^{x}_{\ell} \rangle}^{\mathcal{H}}$ and $\overline{\binom{x}{\ell}}^{\mathcal{H}}$, highlighting some practical relations between such constraints.

COROLLARY 1—$\overline{\binom{x}{\ell}}^{\mathcal{H}}$ DOMINANCE
Given $\lambda_1^{\mathcal{H}} = \overline{\binom{x}{\ell_1}}^{\mathcal{H}}$ *and* $\lambda_2^{\mathcal{H}} = \overline{\binom{x}{\ell_2}}^{\mathcal{H}}$, *if* $\ell_1 \leq \ell_2$ *then*

$$\rho\left(\mathcal{L}_{\lambda_2^{\mathcal{H}}}\right) \leq \rho\left(\mathcal{L}_{\lambda_1^{\mathcal{H}}}\right).$$

COROLLARY 2—$\overline{\langle {}^{x}_{\ell} \rangle}^{\mathcal{H}}$ DOMINANCE
Given $\lambda_1^{\mathcal{H}} = \overline{\langle {}^{x}_{\ell} \rangle}^{\mathcal{H}}$ *and* $\lambda_2^{\mathcal{H}} = \overline{\binom{x}{\ell}}^{\mathcal{H}}$, *then*

$$\rho\left(\mathcal{L}_{\lambda_2^{\mathcal{H}}}\right) \leq \rho\left(\mathcal{L}_{\lambda_1^{\mathcal{H}}}\right).$$

The conclusions drawn from Theorem 1 are theoretical, but its practical applicability lies in the algorithm used to find ρ^{LB} and ρ^{UB}, i.e., lower and upper bounds for the JSR value. Using these bounds we can determine the stability of the corresponding switching systems, as follows:

$$\rho^{LB}\left(\mathcal{L}_{\lambda_2^{\mathcal{H}}}\right) \leq \rho\left(\mathcal{L}_{\lambda_2^{\mathcal{H}}}\right) \leq \rho\left(\mathcal{L}_{\lambda_1^{\mathcal{H}}}\right) \leq \rho^{UB}\left(\mathcal{L}_{\lambda_1^{\mathcal{H}}}\right).$$

Regardless of the algorithm used to find the bounds, if $\lambda_2^{\mathcal{H}} \preceq \lambda_1^{\mathcal{H}}$ and $\rho^{UB}(\mathcal{L}_{\lambda_1^{\mathcal{H}}}) < 1$, the system under $\lambda_2^{\mathcal{H}}$ is switching stable. A similar relation holds for the lower bound.

Theorem 1 can be further extended by relating the joint spectral radius of a single constraint to sets of constraints.

THEOREM 2
Given an arbitrary EWHC $\lambda^{\mathcal{H}}$, *it holds that*

$$\rho\left(\mathcal{L}_{\Lambda^{\mathcal{H}}}\right) \leq \rho\left(\mathcal{L}_{\lambda^{\mathcal{H}}}\right), \quad \forall \Lambda^{\mathcal{H}} \ni \lambda^{\mathcal{H}}.$$

Proof. For an arbitrary EWHC set $\Lambda^{\mathcal{H}}$, its satisfaction set is

$$\mathcal{S}_N\left(\Lambda^{\mathcal{H}}\right) = \bigcap_{\lambda_i^{\mathcal{H}} \in \Lambda^{\mathcal{H}}} \mathcal{S}_N\left(\lambda_i^{\mathcal{H}}\right).$$

Thus, for any $\lambda_i^{\mathcal{H}} \in \Lambda^{\mathcal{H}}$ it holds that

$$\mathcal{S}_N\left(\Lambda^{\mathcal{H}}\right) \subseteq \mathcal{S}_N\left(\lambda^{\mathcal{H}}\right).$$

If a word b is in $\mathcal{S}_N(\Lambda^{\mathcal{H}})$ it also belongs to $\mathcal{S}_N(\lambda^{\mathcal{H}})$. The set of all possible Φ_b is thus included in the set of all possible Φ_a, $a \in \mathcal{S}_N(\lambda^{\mathcal{H}})$. As a consequence it holds that

$$\max_{b \in \mathcal{S}_N(\Lambda^{\mathcal{H}})} \|\Phi_b\|^{1/N} \leq \max_{a \in \mathcal{S}_N(\lambda^{\mathcal{H}})} \|\Phi_a\|^{1/N}, \quad \forall N \in \mathbb{N}_{>}.$$

The theorem follows immediately when $N \to \infty$. □

As in Theorem 1, the more we restrict the execution pattern of the control task with sets of constraints, the lower its JSR will be. Theorem 2 delivers the practical insight that enforcing tighter EWHC to a stable system will *never* destabilise it, as formally stated in the following corollary.

COROLLARY 3
Given an arbitrary EWHC $\lambda^{\mathcal{H}}$*, if* $\rho(\mathcal{L}_{\lambda^{\mathcal{H}}}) < 1$ *then*

$$\rho(\mathcal{L}_{\Lambda^{\mathcal{H}}}) < 1, \ \forall \Lambda^{\mathcal{H}} \ni \lambda^{\mathcal{H}}.$$

6. Evaluation

We apply the lifted dynamics model presented in Section 5 to a representative plant for the process industry, controlled using a PI-controller, sampled with $T = 0.5$ s:

$$\mathcal{P} \ : \ \begin{cases} x_{k+1} &= \begin{bmatrix} 0.606 & 0.304 & 0.076 \\ 0 & 0.606 & 0.304 \\ 0 & 0 & 0.606 \end{bmatrix} x_k + \begin{bmatrix} 0.014 \\ 0.091 \\ 0.394 \end{bmatrix} u_k \\ y_k &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x_k \end{cases}$$

$$\mathcal{C} \ : \ \begin{cases} z_{k+1} &= z_k + 0.359 y_k \\ u_{k+1} &= 0.454 z_k + 0.633 y_k. \end{cases}$$

We analyse the stability of the control systems subject to different $\overline{\binom{x}{\ell}}^{\mathcal{H}}$ constraints. We consider all combinations of strategy (`Kill` or `Skip`) and actuator mode (`Zero` or `Hold`). For each combination, we generate the lifted set $\mathcal{L}_{\lambda^{\mathcal{H}}}$. Its JSR $\rho(\mathcal{L}_{\lambda^{\mathcal{H}}})$ is then approximated using three different algorithms. First, a lower and upper bound of $\rho(\mathcal{L}_{\lambda^{\mathcal{H}}})$ is computed using the `JSR toolbox` [Vankeerberghen et al., 2014]. Then, an upper bound of the JSR is obtained via SOS relaxations, using both the *dense* and *sparse* algorithm from `SparseJSR` [Wang et al., 2021a].

Table 1 displays our results, acquired on an Intel Core i5-8265U@1.60GHz CPU with 8GB RAM. Lower and upper bounds are denoted "LB" and "UB". All upper bounds obtained with `JSR toolbox` was found greater than the ones obtained with SOS, thus omitted from the Table. The symbol "−" means that the SDP solver runs out of memory. The SDP solver in `SparseJSR` uses a second-order method. Thus, a different solver (utilising a first-order method) could reduce memory usage

Table 1. Results obtained for the stable system $\mathcal{P}$, when controlled using $\mathcal{C}$.

		`Kill&Zero`					`Kill&Hold`					`Skip&Zero`					`Skip&Hold`				
$\overline{\binom{x}{\ell}}^{\mathcal{H}}$		JSR		Dense	Sparse		JSR		Dense	Sparse		JSR		Dense	Sparse		JSR		Dense	Sparse	
x	ℓ	LB	UB	UB	UB	×	LB	UB	UB	UB	×	LB	UB	UB	UB	×	LB	UB	UB	UB	×
1	2	0.960	1.094	1.070	1.070	0.86	0.926	1.094	1.029	1.029	0.83	0.922	1.086	**0.924**	**0.924**	5.40	0.958	1.083	**0.958**	**0.958**	4.43
1	3	0.920	1.062	**0.995**	**0.995**	0.83	0.894	1.053	**0.971**	**0.971**	0.77	0.898	1.077	**0.974**	**0.974**	10.5	0.917	1.077	**0.988**	**0.988**	10.4
1	4	0.890	1.038	**0.945**	**0.996**	1.06	0.894	1.021	**0.957**	1.025*	1.25	0.898	1.057	**0.963**	**0.963**	18.2	0.890	1.063	**0.940**	**0.940**	15.9
1	5	0.890	1.011	**0.922**	**0.983**	1.96	0.894	1.011	**0.948**	1.008*	2.25	0.898	1.026	**0.954**	**0.954**	17.6	0.890	1.039	**0.929**	**0.929**	20.8
1	6	0.890	1.012	**0.920**	**0.975**	4.36	0.894	1.016	**0.942**	**0.995**	3.68	0.898	1.016	**0.946**	**0.947**	20.9	0.890	1.023	**0.927**	**0.927**	25.8
2	3	0.983	1.148	1.124	1.124	0.67	0.956	1.152	1.085	1.085	0.80	0.953	1.145	1.034	1.039	4.45	0.982	1.148	1.070	1.076	5.91
2	4	0.960	1.155	1.079	1.079	0.74	0.927	1.160	1.039	1.039	0.86	0.922	1.165	1.033	1.040	23.9	0.958	1.167	1.079	1.086	24.2
2	5	0.939	1.156	1.039	1.142	2.09	0.905	1.156	1.002	1.105	1.58	0.898	1.186	**0.999**	1.005	77.8	0.937	1.182	1.038	1.043	58.1
2	6	0.920	1.150	1.007	1.096	12.3	0.903	1.145	**0.974**	1.080	19.2	0.907	1.184	–	1.007	–	0.917	1.182	–	**0.991**	–
3	4	0.990	1.186	1.133	1.133	0.76	0.967	1.192	1.098	1.098	1.69	0.967	1.177	1.072	1.082	6.59	0.990	1.191	1.106	1.117	5.02
3	5	0.975	1.210	1.109	1.109	0.77	0.946	1.215	1.071	1.071	1.74	0.942	1.234	1.071	1.080	34.3	0.975	1.233	1.116	1.125	35.2
3	6	0.960	1.247	1.082	1.227	2.61	0.928	1.252	1.043	1.182	3.25	0.921	1.246	–	1.118	–	0.959	1.242	–	1.072	–
4	5	0.994	1.198	1.130	1.130	1.06	0.976	1.206	1.099	1.099	0.82	0.974	1.189	1.122	1.134	5.43	0.993	1.121	1.088	1.100	5.16
4	6	0.983	1.260	1.120	1.120	0.68	0.957	1.267	1.084	1.084	0.64	0.953	1.267	–	1.143	–	0.983	1.265	–	1.100	–

at the cost of potential accuracy loss. Bold values represent stable systems under their corresponding EWHC, strategy, and actuator mode. Starred values represent stable systems inferred from Corollary 1. The `JSR toolbox` provides an accurate lower bound and a coarse upper bound. In contrast, the dense SOS method finds a better upper bound but takes more time. We compare the time to run both SOS methods, indicating with "$\times$" the speedup factor to obtain the sparse bound w.r.t. the dense.

All the upper bounds computed by `JSR toolbox` are greater than 1, while all lower bounds are *below* 1, thus we cannot draw any conclusion using the `JSR toolbox`. For all EWHC, $\overline{\binom{x}{\ell}}^{\mathcal{H}}$ where $x = 1$ and $2 < \ell \leq 6$ the SOS upper bounds allow us to infer that the system is stable for all combinations of strategy and actuator mode, and also for $\ell = 2$ under the `Skip` strategy. From Theorem 1, the stability will hold also for all constraints that are harder to satisfy; in particular, Corollary 1 implies stability for all $\overline{\binom{x}{\ell}}^{\mathcal{H}}$ with $x = 1$ and $\ell > 6$. The speedup ratio is growing when ℓ increases, yielding a particularly high benefit of exploiting sparsity for `Skip&Zero`.

7. Conclusion

This paper proposes a switching stability analysis framework for LTI systems with arbitrary weakly-hard constraints, extending the weakly-hard model and providing an analytic stability bound. The analysis allows us to assess whether computational errors (present in industrial controllers) affect the stability of the controlled systems. Future work will focus on the performance loss due to the presence of deadline misses following the extended weakly-hard model.

Acknowledgements

Nils Vreman and Martina Maggio are members of the ELLIIT Strategic Research Area at Lund University. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Number 871259 (ADMORPH project). This publication reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

References

Akesson, B., M. Nasri, G. Nelissen, S. Altmeyer, and R. Davis (2020). "An empirical survey-based study into industry practice in real-time systems". In: *Real-Time Systems Symposium*.

Bernat, G., A. Burns, and A. Liamosi (2001). "Weakly hard real-time systems". *IEEE Transactions on Computers* **50**:4, pp. 308–321. DOI: 10.1109/12.919277.

Cervin, A. (2005). "Analysis of overrun strategies in periodic control tasks". *IFAC Proceedings Volumes* **38**:1. 16th IFAC World Congress, pp. 219–224. ISSN: 1474-6670. DOI: 10.3182/20050703-6-CZ-1902.01076.

Choi, H., H. Kim, and Q. Zhu (2019). "Job-class-level fixed priority scheduling of weakly-hard real-time systems". In: *Real-Time and Embedded Technology and Applications Symposium*.

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

Gaukler, M., T. Rheinfels, P. Ulbrich, and G. Roppenecker (2019). "Convergence rate abstractions for weakly-hard real-time control". *arXiv preprint arXiv:1912.09871*.

Hertneck, M., S. Linsenmayer, and F. Allgöwer (2021). "Efficient stability analysis approaches for nonlinear weakly-hard real-time control systems". *Automatica*.

Huang, C., K.-C. Chang, C.-W. Lin, and Q. Zhu (2020). "Saw: a tool for safety analysis of weakly-hard systems". In: *International Conference on Computer Aided Verification*. Springer.

Huang, C., W. Li, and Q. Zhu (2019). "Formal verification of weakly-hard systems". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*.

Kozyakin, V. (2014). "The berger–wang formula for the markovian joint spectral radius". *Linear Algebra and its Applications* **448**.

Lasserre, J. (2001). "Global optimization with polynomials and the problem of moments". *SIAM Journal on optimization* **11**:3.

Liang, H., Z. Wang, R. Jiao, and Q. Zhu (2020). "Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees". In: *Conference On Computer Aided Design*.

Liang, H., Z. Wang, D. Roy, S. Dey, S. Chakraborty, and Q. Zhu (2019). "Security-driven codesign with weakly-hard constraints for real-time embedded systems". In: *International Conference on Computer Design*.

Linsenmayer, S. and F. Allgower (2017). "Stabilization of networked control systems with weakly hard real-time dropout description". In: *56th IEEE Conference on Decision and Control (CDC)*, pp. 4765–4770.

Linsenmayer, S., M. Hertneck, and F. Allgower (2020). "Linear weakly hard real-time control systems: time-and event-triggered stabilization". *IEEE Transactions on Automatic Control*.

Maggio, M., A. Hamann, E. Mayer-John, and D. Ziegenbein (2020). "Control-system stability under consecutive deadline misses constraints". In: *32nd Euromicro Conference on Real-Time Systems (ECRTS)*. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Parrilo, P. and A. Jadbabaie (2008). "Approximation of the joint spectral radius using sum of squares". *Linear Algebra and its Applications*.

Pazzaglia, P., A. Hamann, D. Ziegenbein, and M. Maggio (2021). "Adaptive design of real-time control systems subject to sporadic overruns". In: *Design, Automation & Test in Europe Conference Exhibition*.

Pazzaglia, P., C. Mandrioli, M. Maggio, and A. Cervin (2019). "DMAC: Deadline-Miss-Aware Control". In: *31st Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 1:1–1:24. ISBN: 978-3-95977-110-8.

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

Rota, G. and W. Strang (1960). "A note on the joint spectral radius".

Schenato, L. (2009). "To zero or to hold control inputs with lossy links?" *IEEE Transactions on Automatic Control* **54**:5, pp. 1093–1099.

Vankeerberghen, G., J. Hendrickx, and R. Jungers (2014). "Jsr: a toolbox to compute the joint spectral radius". In: *International Conference on Hybrid Systems Computation and Control*.

Vreman, N., A. Cervin, and M. Maggio (2021). "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.15.

Vreman, N., R. Pates, and M. Maggio (2022). "Weaklyhard.jl: scalable analysis of weakly-hard constraints". In: *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 228–240. DOI: 10.1109/RTAS54340.2022.00026.

Wang, J., M. Maggio, and V. Magron (2021a). "SparseJSR: A Fast Algorithm to Compute Joint Spectral Radius via Sparse SOS Decompositions". *American Control Conference*.

Wang, J., V. Magron, and J.-B. Lasserre (2021b). "TSSOS: A Moment-SOS hierarchy that exploits term sparsity". *SIAM Journal on Optimization* **31**:1.

Xu, W., Z. Hammadeh, A. Kröller, R. Ernst, and S. Quinton (2015). "Improved deadline miss models for real-time systems using typical worst-case analysis". In: *Euromicro Conference on Real-Time Systems*.

Xu, X. and B. Acikmese (2020). "Approximation of the constrained joint spectral radius via algebraic lifting". *Transactions on Automatic Control*.

Xu, X. and Y. Hong (2012). "Matrix expression and reachability analysis of finite automata". *Journal of Control Theory and Applications* **10**:2.

Paper V

Stochastic Analysis of Control Systems Subject to Communication and Computation Faults

Nils Vreman **Martina Maggio**

Abstract

Control theory allows one to design controllers that are robust to external disturbances, model simplification, and modelling inaccuracy. Researchers have investigated whether the robustness carries on to the controller's digital implementation, mostly looking at how the controller reacts to either communication or computational problems. Communication problems are typically modelled using random variables (i.e., estimating the probability that a fault will occur during a transmission), while computational problems are modelled using deterministic guarantees on the number of deadlines that the control task has to meet. These fault models allow the engineer to both design robust controllers and assess the controllers' behaviour in the presence of isolated faults. Despite being very relevant for the real-world implementations of control system, the question of what happens when these faults occur simultaneously does not yet have a proper answer. In this paper, we answer this question in the stochastic setting, using the theory of Markov Jump Linear Systems to provide stability contracts with *almost sure* guarantees of convergence. We apply our method to two case studies from the recent literature and show their robustness to a comprehensive set of faults.

Submitted to ACM SIGBED International Conference on Embedded Software (2023).

1. Introduction

Two important objectives in the design of control systems are guaranteeing robustness to disturbances and modelling inaccuracy [Åström and Wittenmark, 1997], and the joint verification of the computer program implementing the controller logic, together with the physical system this program acts on [Bohrer et al., 2018]. The outcome of the verification process is a certificate of correctness for the cyber-physical system that comprises both the controller and the physical plant it controls.

However, during its actual execution, the controller operation can be affected by faults such as computational delays and communication loss. Researchers have been trying to quantify how much of the inherent controller robustness carries over to tolerate communication [Ahrendts et al., 2018; Linsenmayer and Allgower, 2017; Yang and Ozay, 2021] and computational [Pazzaglia et al., 2018; Maggio et al., 2020; Hobbs et al., 2022] faults. The main appeal with these results is the ability to guarantee properties of the closed-loop systems in worst-case conditions.

In real-world controller implementations, worst-case conditions are rare. The results obtained to certify worst-case conditions may be exceedingly conservative under normal operation [Vreman et al., 2021]. Furthermore, the process of obtaining computational models (such as the weakly-hard [Bernat et al., 2001] task model) that enable these analyses is still complex [Sun and Natale, 2017], restricting the applicability of the controller analyses. On the contrary, typical fault models are probabilistic. If some risk is tolerated, or if the worst-case is extremely rare, soft real-time task models [Buttazzo et al., 2005] (i.e., *probabilistic* or *stochastic*) can significantly improve typical-case performance analysis. These probabilistic models aim to optimise the average-case performance rather than the worst-case robustness, and can also be used to provide stochastic safety guarantees.

There exists a vast literature on stochastic stability for control systems, including [Fang and Loparo, 2002; Liberzon, 2014; Blair Jr. and Sworder, 1975; Lincoln and Cervin, 2002; Bolzern et al., 2010; Åström, 1970]. These results are typically providing guarantees on the safe operation of a control system in the presence of stochastic disturbance signals, rather than to guarantee the safe operation of a control system in the presence of computational problems. Moreover, the literature on fault tolerance typically answers questions such as when is the first fault occurring [Safari et al., 2022]. However, predicating over the safety of the control system in the presence of faults should also take into account that multiple components can fail at the same time. For example, a networked control system can experience a channel dropout *simultaneously* with the controller code stalling and thus not completing its execution before its deadline.

Although some literature indicate that tolerating packet losses is enough for networked control systems to function also in the presence of other fault types [Kauer et al., 2014; Ghosh et al., 2018; Horssen et al., 2016; Ling and Lemmon, 2002; Linsenmayer and Allgower, 2017], this can only be true for static controllers, e.g., LQ-regulators. For such controllers, a lost packet is equivalent to a missed deadline

if the controller waits indefinitely for a sensor packet to arrive. Even with the restriction of using a static controller, the assumption that the controller would wait for sensor data indefinitely is both conservative and unrealistic.

In this paper we aim to resolve the misconception that packet losses in networked control system can be used to analyse control deadline overruns (and vice versa). We formulate the problem of analysing a control system in the presence of *simultaneous* failures of three different types: (i) packet losses on the sensor channel, (ii) computational overruns of the control task, and (iii) packet losses on the actuator channel. In solving this problem, we aim at bringing the control analysis one step closer to the implementation of control tasks, considering actual control skeletons that include, among other things, timeouts for communication channels.

To analyse the simultaneous presence of faults in computational units and communication channels, this paper casts the problem into the formalism of Markov Jump Linear Systems [Costa et al., 2005] and provides a stochastic analysis of the controller behaviour. Specifically, we provide the following contributions:

- We compile a model of what happens to the control system when the different faults are experienced. This model includes both the discrete state (which encodes whether data transmissions and control computation have been successful) and the dynamical state of the physical part of the system (which describes the quantities that are affected by the controller execution both in the controller itself and in the physical world).
- We leverage the literature on stochastic control to provide a probabilistic analysis of control systems subject to simultaneous communication and computation faults. The analysis aims to provide a certificate of *mean square stability*, i.e., the simultaneous convergence of the average value of the state vector to a precise point, and of its covariance (and hence standard deviation) to zero.
- We apply the analysis to two different case studies taken from the literature, showing how resilient their controllers are to faults that may occur during the lifetime and execution of the controller.

The rest of this paper is outlined as follows. Section 2 contains the necessary background and a more precise problem statement. In Section 3 we propose an analysis of control systems subject to *both* packet losses and computational overruns. In Section 4 we evaluate the analysis on two different case studies taken from the literature, and show that we can assess the (stochastic) stability of the controlled systems under a variety of simultaneous faults. Section 5 summarises the related literature and Section 6 concludes the paper.

2. Problem Formulation

This section provides the necessary background and introduces the cyber-physical system models used in the remainder of the paper. In particular, Section 2.1 provides

a brief overview of the models used in the design of linear control systems and Section 2.2 discusses the implementation choices made in the realisation of the controller code, and the faults that can be experienced by the controller. Finally, Section 2.3 formalises the problem addressed in this paper.

2.1 Control System Synthesis

The objective of a feedback control system is to make a physical process (denoted *plant*) behave according to some predetermined requirements. Such requirements generally include stabilising the plant, rejecting disturbances, and tracking a desired trajectory. Stability is essential to guarantee that physical quantities stay bounded.

In their most general form, the plant dynamics are continuous-time and non-linear. However, for control analysis and synthesis purposes [Åström and Wittenmark, 1997], a simpler model of the plant is generally devised and discretised, generally obtaining a discrete-time Linear Time-Invariant (LTI) state-space system. The system typically takes the form

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

In the equation, k counts the discrete number of samples elapsed since system startup, $x_k \in \mathbb{R}^{n_x}$ is the state vector, $u_k \in \mathbb{R}^{n_u}$ contains the control commands used to affect the plant, and $y_k \in \mathbb{R}^{n_y}$ is the sensor measurements. The plant dynamics is encoded in the matrices $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$, and $D \in \mathbb{R}^{n_y \times n_u}$. The eigenvalues of A, determine if the system is inherently stable ($\max|\mathrm{eig}\,(A)| < 1$) or not.

To satisfy the requirements, a *controller* is designed for and implemented on digital hardware. Generally, controllers are designed and implemented following the Logical Execution Time (LET) paradigm [Henzinger et al., 2003], i.e., the sensor messages are received at the beginning of the control computation and the actuator messages are sent at the end of the control period.[1] Similarly to plants, controllers are generally described using discrete-time, LTI state-space systems

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

Here, $z_k \in \mathbb{R}^{n_z}$ is the controller's internal state vector. The controller dynamics is described by the matrices $F \in \mathbb{R}^{n_z \times n_z}$, $G \in \mathbb{R}^{n_z \times n_y}$, $H \in \mathbb{R}^{n_u \times n_y}$, and $K \in \mathbb{R}^{n_u \times n_z}$.[2]

Combining the dynamical models of the plant and controller we obtain the *closed-loop system* $\mathcal{S}_{\mathrm{cl}}$

$$\mathcal{S}_{\mathrm{cl}}:\ \tilde{x}_{k+1} = \Phi\,\tilde{x}_k. \tag{3}$$

[1] The LET paradigm increases timing predictability and reduces jitter at the cost of introducing a one-step delay in the control signal, i.e., u_{k+1}.

[2] A controller is *stateless* when $n_z = 0$ and *stateful* otherwise, i.e., $n_z > 0$. Stateless controllers can always be written as $\mathcal{C}: u_{k+1} = K\,y_k$.

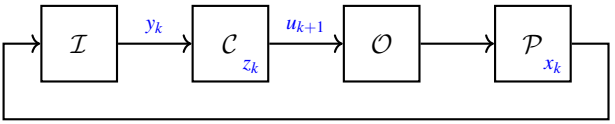


Figure 1. Block diagram representing the interconnection of different components in a control system. The controller $\mathcal{C}$ receives input from the sensor $\mathcal{I}$ and sends data to the actuator $\mathcal{O}$, which in turn acts on the plant $\mathcal{P}$.

Here, $\tilde{x}_k$ is the closed-loop system's state vector (with initial state $\tilde{x}_0$) and Φ encodes the closed-loop system's dynamics. For the plant and controller models used in this paper, the closed-loop state vector can be reduced down to $\tilde{x}_k = [x_k^{\mathrm{T}}, z_k^{\mathrm{T}}, u_k^{\mathrm{T}}]^{\mathrm{T}}$, where T is the transpose operator. The nominal behaviour of $\mathcal{S}_{\mathrm{cl}}$ can then be described by

$$\underbrace{\begin{bmatrix} x_{k+1} \\ z_{k+1} \\ u_{k+1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & 0 & B \\ GC & F & GD \\ KC & H & KD \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix}}_{\tilde{x}_k}. \tag{4}$$

To assess whether a closed-loop system is stable under nominal conditions, it is sufficient to check whether *all* eigenvalues of Φ lie inside the unit disc [Åström and Wittenmark, 1997]. Denoting with $\rho(\Phi)$ the largest absolute magnitude of an eigenvalue of Φ, then $\mathcal{S}_{\mathrm{cl}}$ is stable if and only if

$$\rho(\Phi) = \max |\mathrm{eig}(\Phi)| < 1. \tag{5}$$

A schematic implementation of a closed-loop real-time control system $\mathcal{S}_{\mathrm{cl}}$ can be seen in Figure 1. Starting from the *sensors* $\mathcal{I}$, the plant is sampled at discrete time instants k. The controller $\mathcal{C}$ then polls the sensor channel for the latest sampled measurement signal to use in the calculation of the new control command u_k. When the new control command is computed, the controller stores it in memory before sending it to the *actuator* $\mathcal{O}$ at the beginning of the next control iteration. Finally, the actuator acts upon the plant $\mathcal{P}$, based on the command it received from the controller.

2.2 Fault Model

As anticipated, we aim at devising a stochastic analysis that determines the stability of the closed-loop system in the presence of faults. We assume that three components in the real-time control system can experience faults (possibly simultaneously) (i) the *sensor channel*, that transmits information between the sensor $\mathcal{I}$ and controller $\mathcal{C}$, (ii) the *control task*, that executes the algorithm of the controller $\mathcal{C}$, and (iii) the *actuator channel*, that allows the controller $\mathcal{C}$ to communicate with the actuator $\mathcal{O}$. We provide a brief review of what can cause problems for both the controller and the input/output (IO) channels, together with some common implementation details.

Control Task and Overruns Controllers $\mathcal{C}$ should periodically calculate a control signal based on (2). They are generally implemented in *control tasks*, i.e., periodic tasks with implicit deadlines. A typical implementation is the following.

```
1 while True:
2     y = read_sensor_ch()
3     u, z = compute_control(y, z)
4     sleep_until(next_activation)
5     send_actuator_ch(u)
```

Listing 3.1. Typical control algorithm execution.

The code in Listing 3.1 performs the following operations (i) it samples the current plant measurements in `y` using the sensor $\mathcal{I}$ (via the function `read_sensor_ch`), (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 $\mathcal{O}$ (via `send_actuator_ch`).

For the control task, each iteration of the loop in Listing 3.1 is a new *job*, and the k-th iteration corresponds to the job j_k. The control job j_k is *released* at time $a_k = kT$, where T is the *period* of the control task. The objective of each job is to complete its execution before its corresponding *deadline* $d_k = a_k + T = (k+1)T$. 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. Overruns can be caused by many different factors, like preemption from higher priority tasks and interrupts [Stankovic et al., 1995], timeouts due to long wait times on the sensor channel [Ohlin et al., 2006], and cache misses that introduce delays in accessing the controller's stored variables [Wang et al., 2012].

Regardless of what caused the overrun, the scheduler needs to react. In the literature [Cervin, 2005], mainly three simple *deadline overrun strategies* have been considered (i) `Kill` – i.e., *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) `Skip` – i.e., letting the job continue its execution, *skipping* the subsequent job releases, until the current one has completed its execution, or (iii) `Queue` – i.e., 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. The `Queue` strategy has been shown to create chain effects which can severely damage control systems [Cervin, 2005; Maggio et al., 2020], hence in this paper we only consider `Kill` and `Skip` as viable strategies.

Sensor Channel Dropouts Control systems rely on communication interfaces between the sensors/actuator and the controller code itself (i.e., the sensor and actuator channels). A significant amount of research, including [Ling and Lemmon, 2002; Linsenmayer and Allgower, 2017; Kauer et al., 2014; Goswami et al., 2014], analysed the problem of control system's stability and performance when subject to sensor packet losses. Losing packets over the sensor channel can lead to the controller not updating the control command, using old measurement data, or even missing job deadlines due to prolonged waiting times. In these cases, the time in between two sampling instants is time-varying, and control design strategies can be applied to optimise the control system performance [Ghosh et al., 2018; Schinkel et al., 2002]. However, this work does not take into account that packet losses could be combined with deadline overruns.

Furthermore, in the code snippet shown in Listing 3.1, there is no indication of how the control task reacts to sensor packet losses. In classical controller implementations, if no sensor packet is received within a given time limit (typically a fraction of the deadline), the function `read_sensor_ch` *times out*, and the control algorithm can perform one of the following two actions (i) *continue* its execution, without updating the value of `y`, thus using the previous received sensor value, or (ii) *avoid* executing the remaining instructions, terminating the computation early.

The choice of continuing or avoiding is highly dependent on the system dynamics. Assuming that packet losses are relatively uncommon and the control period is typically short, using continue is advantageous. In fact, if a packet is lost, it is still likely that the previous value reported by the sensor is a reasonable approximation of the physical environment. We emphasise that continuing the execution is equivalent to the case when the controller does not directly poll the sensor channel, but rather reads the most recently received sensor value from memory (where it was stored by a receiver task). On the other hand, avoiding the execution of the remaining instructions also prevents the controller from evolving its state `z` and control signal `u` in a possibly unsafe direction. Listings 3.2 and 3.3 show more realistic versions of Listing 3.1 for the continue and avoid cases.

```
1 while True:
2     y, tout_triggered = read_sensor_ch(tout_seconds)
3     if tout_triggered:
4         y = y_old
5     u, z = compute_control(y, z)
6     y_old = y
7     sleep_until(next_activation)
8     send_actuator_ch(u)
```

Listing 3.2. Control code execution when the control computation is continued if the sensor reading function `read_sensor_ch` results in a timeout.

```
while True:
    y, tout_triggered = read_sensor_ch(tout_seconds)
    if not tout_triggered:
        u, z = compute_control(y, z)
    sleep_until(next_activation)
    send_actuator_ch(u)
```

Listing 3.3. Control code execution when the control computation is avoided if the sensor reading function `read_sensor_ch` results in a timeout.

In Listings 3.2 and 3.3, when `tout_seconds` time units have passed without completion, the receive function `read_sensor_ch` sets the variable `tout_triggered` to true. In Listing 3.2, the control algorithm continues executing its instructions using the old sensor value `y_old`, i.e., the controller's internal state `z` will be updated and a new control command `u` will be sent to the actuators. In Listing 3.3, the control algorithm will not be run, i.e., the internal state `z` and the control command `u` are kept constant.

Actuator Channel Dropouts Packet losses on the actuator channel have not been studied as thoroughly as their sensor counterpart. The reason likely comes from the fact that the actuator response is typically hardware-dependant and difficult to detect and compensate in the control algorithm. If the actuator does not receive a new control command when it is expecting one, it defaults to a value dependent on the *actuation mode*. The actuation mode has received more attention, with different degradation or stabilising actuation policies being proposed [Ma et al., 2018]. However, the most common approaches involve either `Hold`ing the last received control command (i.e., $u_{k+1} = u_k$) or `Zero`ing the output (i.e., $u_{k+1} = 0$). Similarly to choosing between continue and avoid, the choice of actuation mode is non-trivial and generally depend on the control system dynamic [Schenato, 2009; Vreman et al., 2021].

2.3 Problem Formulation

Many different analysis frameworks have been proposed to evaluate the computational robustness to packet loss and deadline overruns of controllers [Ghosh et al., 2018; Maggio et al., 2020; Linsenmayer and Allgower, 2017; Donkers et al., 2011]. However, these analyses have two major shortcomings: (i) they are developed in isolation, and do not combine the presence of potential problems both in the computation and in the IO channels, and (ii) they rely on knowledge about the occurrence of events like deadline overruns or packet losses. In fact, recent works on computational overruns [Maggio et al., 2020; Linsenmayer and Allgower, 2017] rely on the weakly-hard task model [Bernat et al., 2001] to constraint the sequence of deadline overruns; and recent work like [Ghosh et al., 2018] are on the contrary working on the assumption that packet losses are detected and counteracted at the control level. However, typical methods for deriving bounds on both packet losses

and deadline overruns are probabilistic; for example, estimating the probability that a specific task in a system will overrun its deadline when a particular scheduling algorithm is employed [Chen et al., 2019; Brüggen et al., 2021].

This paper aims at providing a control-theoretical analysis for how the closed-loop system robustness is affected by stochastic packet losses (on sensor and actuator channels), deadline overruns, and a combination thereof. Furthermore, we want to devise an analysis method that benefits from the state-of-the-art results on fault occurrence estimation in real-time systems implementations [Chen et al., 2019; Brüggen et al., 2021]. We assume to receive, as input, probabilities p_c, p_s, and p_a. These represent respectively the probability of the control task missing its deadline, the probability of a failure on the sensor channel and the probability of a packet loss on the actuator channel. The analysis provides – as output – a certificate that specifies that the system does or does not satisfy a stochastic stability requirement. If the certificate verifies the stochastic stability of the closed-loop system, then the average value $\mathbb{E}[\tilde{x}_k]$ of the system state $\tilde{x}_k$, introduced in Equation (3), converges to a given value and its standard deviation converges to zero. This allows us to validate the control system implementation behaviour in the presence of undesirable faults.

3. Analysis

Our analysis of the closed-loop system subject to IO channel dropouts and deadline overruns is based on the theory of Markov Jump Linear Systems [Costa et al., 2005]. These systems combine the dynamics of the closed-loop system and the transition probabilities of faults and errors. In Section 3.1 we provide some preliminary definitions and in Section 3.2 we derive the control system dynamics when (possibly simultaneously) deadline overruns, sensor data loss, and actuator data loss occur. In Section 3.3 we present the Markov chain that describes the probabilistic evolution of the discrete state of the system. Finally, in Section 3.4 we summarise and apply the Markov Jump Linear Systems theory to the closed-loop system, obtaining the stochastic stability certificates. Note that the concept of *state* differs between Markov and control theory. With the word *state* we denote the dynamical system's state vector $\tilde{x}_k$, whilst the information encoded by a sequence of events including IO channel dropouts and computational overruns is referred to as the *discrete state*.

3.1 Event Outcomes

Equation (4) presented the dynamical model of the closed-loop system and the closed-loop state matrix Φ in nominal conditions, i.e., in the absence of faults. However, as discussed in Section 2, the system dynamics are heavily impacted by whether the controller misses a control computation or experiences a packet loss on a communication channel. To analyse the system dynamics, we define the *outcome set* Σ for the transmission on IO channels and the computation of the controller.

DEFINITION 1—IO CHANNEL OUTCOME SETS
We denote the set of outcomes that each packet on the sensor and actuator channels can experience by $\Sigma(\mathcal{I}) = \Sigma(\mathcal{O}) = \{F, T\}$*:*

- F*: represents a lost packet, and*
- T*: represents a successfully delivered packet.*

Trivially, the outcome of each packet transmission is a binary event where either: (i) the packet is successfully received (T), or (ii) the packet is lost along its route (F). Since the contents of the packet (e.g., measurement data from the sensors or control commands to the actuators) is irrelevant to whether the packet is lost or not, the same outcome notation is used for both $\Sigma(\mathcal{I})$ and $\Sigma(\mathcal{O})$.

Unlike the IO channels, the outcome of a control job's computation is not necessarily a binary event. In particular, the overrun strategy employed by the scheduler determines the outcome set. We now define the control task outcome sets (for one execution interval, i.e., for one *control period*) both for the `Kill` and for the `Skip` strategy.

DEFINITION 2—COMPUTATIONAL OUTCOME SET - `Kill`
We denote the set of outcomes that a control job can experience in each control period when the scheduler adopts the `Kill` *strategy by* $\Sigma(\mathcal{C}^K) = \{M, H\}$.

- M*: a job is released, but no job is completed,*
- H*: a job is both released and completed.*

DEFINITION 3—COMPUTATIONAL OUTCOME SET - `Skip`
We denote the set of outcomes that a control job can experience in each control period when the scheduler adopts the `Skip` *strategy by* $\Sigma(\mathcal{C}^S) = \{M, N, H, R\}$.

- M*: a new control job is released but no job is completed,*
- N*: no job is either released or completed,*
- H*: a new control job is both released and completed,*
- R*: no job is released, but a job (that was released in a previous period) is completed.*

For both `Kill` and `Skip`, each control period contains *at most* one activated and one completed job. The main difference comes from the `Kill` strategy terminating every job that overrun its corresponding deadline, i.e., each control period contains a new control job being released and activated. Thus, the outcome set $\Sigma(\mathcal{C}^{\mathrm{K}})$ consist of only two outcomes: a job being completed or a job not being completed. On the other hand, the `Skip` strategy encompasses more diverse outcomes. For instance,

the outcomes M and N both encode an overrun deadline; but M represent the start of a control computation while N correspond to its continuation. Finally, R is used to identify the *late completion* of a job, i.e., a *recovery hit*. The occurrence of R and N impose constraints on the outcome ordering. We note that if we use the **Skip** overrun strategy, there is a natural valid order between the job outcomes. The following constraint enforces that a sequence of job outcomes is valid, i.e., it can be produced by a control task.

CONSTRAINT 1
For a sequence of job outcomes under the **Skip** *overrun strategy, it holds that:*

- *both* M *and* H *are restricted to directly follow an* H *or* R*,*
- *an* N *can only follow an* M*, and*
- *an* R *may only directly follow an* N *or* M*.*

3.2 Closed-Loop System Dynamics

From Definitions 1-3, the closed-loop system dynamics' evolution in time can be fully derived as (with initial state $\tilde{x}_0$):

$$\tilde{x}_{k+1} = \Phi_{sca}\tilde{x}_k. \tag{6}$$

As for Equation (3), $\tilde{x}_k$ is the closed-loop state vector and Φ_{sca} is the closed-loop system matrix. The system dynamics does however depend on the outcome realisations, i.e., $s \in \Sigma(\mathcal{I})$, $a \in \Sigma(\mathcal{O})$, and $c \in \Sigma(\mathcal{C}^{\bullet})$ (where $\bullet$ is either K or S). If a fault occurs, the closed-loop system's evolution will deviate from the nominal behaviour. As an example, the closed-loop system matrix $\Phi_{\mathtt{THT}}$ would correspond to a control job receiving the sensor message, completing its algorithm execution in the same period as it was released, and the actuators would successfully receive the new control command. Trivially, $\Phi_{\mathtt{THT}}$ correspond to the nominal behaviour from Equation (4). Instead, if an actuator packet would be lost, the control command sent to the plant would depend on the actuator mode. Therefore, the closed-loop system would evolve according to $\Phi_{\mathtt{THF}}$.

Note that in each control period the system experiences a new realisation of the outcome sets, i.e., the closed-loop system matrix Φ_{sca} can switch every control period. The resulting dynamics is generally called a *switching system*. Since the closed-loop dynamics is no longer consistent between periods, the stability criterion presented in (5) cannot be used to assess the stability of the system, and we need to resort to a probabilistic stability result, presented in Section 3.4.

As discussed in Section 2, the system dynamics change with respect to (i) the overrun strategy, (ii) the actuation mode, and (iii) the choice of timeout strategy. In this paper, we focus on the stability analysis for the continue case, as the possible *outcome configurations* (i.e., combinations of s, c, and a) for the avoid case are included in the set of outcome configurations for continue, making the analysis

simpler in the avoid case.[3] In the continue case, we provide a stability analysis for both actuation modes (i.e., `Zero` or `Hold`) and both overrun strategies (i.e., `Kill` or `Skip`). The difference between `Zero` and `Hold` result in variations of the closed-loop matrices, and is encoded using the symbol $\Delta_{\mathcal{O}}$, described later. However, the difference between using `Kill` and `Skip` fundamentally changes the structure of the outcome set of the control jobs changes. Hence, we need to analyse the `Kill` and `Skip` cases separately.

`Kill`: When a fault is experienced (deadline overrun or packet loss), the physical state of the plant x_k continue its normal evolution. However, the closed-loop will not behave according to its design specifications. In case j_k overruns its deadline, its execution is terminated and the controller's states are rolled back, i.e., $z_{k+1} = z_k$. Furthermore, when j_k overruns its deadline *or* the actuator channel experiences a packet loss, the control command defaults to a value that depends on the actuation mode, i.e., $u_{k+1} = u_k$ if the actuation mode is `Hold`, and $u_{k+1} = 0$ if the actuation mode is `Zero`. On the contrary, if a sensor packet is not received, the control algorithm computes a new control command and updates the controller's internal state, using outdated sensor measurements.

To properly describe the closed-loop behaviour under continue and `Kill`, we define the closed-loop state vector $\hat{x}_k^{\mathtt{K}} = [x_k^{\mathsf{T}}, z_k^{\mathsf{T}}, u_k^{\mathsf{T}}, \hat{y}_{k-1}^{\mathsf{T}}]^{\mathsf{T}}$. The introduced auxiliary state $\hat{y}_{k-1}$ records the old sensor value, that is used if `read_sensor_ch` times out (see Listing 3.2). The set of closed-loop matrices describing the system behaviour for the different outcome configurations is then

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ KC & H & KD & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{THT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{THF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{TMT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{TMF}}}$$

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{FHT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{FHF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{FMT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{K}}_{\mathtt{FMF}}}. \tag{7}$$

Each outcome (s, c, and a) has 2 possible configurations in the `Kill` case. Hence, there are $2^3 = 8$ possible *sca*, where $s \in \Sigma(\mathcal{I})$, $c \in \Sigma(\mathcal{C}^{\mathtt{K}})$, and $a \in \Sigma(\mathcal{O})$. The symbol $\Delta_{\mathcal{O}}$ in Equation (7) is used to distinguish the actuator mode, and is either $\Delta_{\mathcal{O}} = I$ for the `Hold` mode or $\Delta_{\mathcal{O}} = 0$ for the `Zero` mode. The variable $\hat{y}_k$ is not updated for $s = \mathtt{F}$, i.e., $\hat{y}_k = \hat{y}_{k-1}$. Additionally, some matrices are identical (e.g., $\Phi^{\mathtt{K}}_{\mathtt{TMT}} = \Phi^{\mathtt{K}}_{\mathtt{TMF}}$ and $\Phi^{\mathtt{K}}_{\mathtt{FMT}} = \Phi^{\mathtt{K}}_{\mathtt{FMF}}$), highlighting for example that in case of `Kill` when the computation

[3] Additionally, *continue* also cover the case where the controller reads sensor data directly from a memory register instead of polling the sensor channel.

does not complete, receiving or not receiving the actuator signal is irrelevant for the system evolution.

Skip: Similarly to the `Kill` case, when the overrun mode is set to `Skip`, the physical states x_k continue their evolution. However, in contrast to the `Kill` overrun strategy, when a job j_k overruns its deadline, the job is allowed to continue its execution, and no subsequent jobs are released until j_k completes its execution.

As an example, assume that job j_k, released at time a_k, finishes its execution at time $f_k = a_k + 3.7T$. In this example, three subsequent jobs are skipped. During the three periods in which j_k is pending completion, no new job is released and the actuator outputs a control command that is in line with the actuation mode, either `Hold` or `Zero`. When j_k completes its execution, the controller state is updated and the control command is computed using the sensor value that was retrieved at time a_k, i.e., depending on whether the sensor packet at time a_k was received or not. In this case, the new control signal is sent to the actuator at the end of the control period, at time instant $a_k + 4T$.

The closed-loop state vector for the continue and `Kill` case can be reused to describe the system evolution of the continue and `Skip` case, i.e., $\tilde{x}_k^{\mathtt{S}} = [x_k^{\mathrm{T}}, z_k^{\mathrm{T}}, u_k^{\mathrm{T}}, \hat{y}_{k-1}^{\mathrm{T}}]^{\mathrm{T}}$. Intuitively, there are 16 possible outcome configurations since $\Sigma(\mathcal{C}^{\mathtt{S}})$ contains four outcomes rather than two, i.e., $2^2 \cdot 4 = 16$. The symbol $\Delta_{\mathcal{O}}$ is again used to indicate the chosen actuation mode. The closed-loop matrices are

$$
\begin{aligned}
&\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ KC & H & KD & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{THT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{THF}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{TMT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{TMF}}} \\
&\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{TNT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{TNF}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FNT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FNF}}} \\
&\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{TRT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FRT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{TRF}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FRF}}} \\
&\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FHT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FHF}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FMT}}}
\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{O}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Phi^{\mathtt{S}}_{\mathtt{FMF}}}
\end{aligned}
\tag{8}
$$

When $c = \mathtt{R}$ or $c = \mathtt{N}$, the sensor packet's outcome is irrelevant for the system dynamics. This follows since the control task does not poll the sensor channel for packets if it is still executing the body of the control algorithm (unless its outcome is M, since it is the first period that experiences an overrun). Again, note that many matrices are identical:

$$\Phi^{\mathtt{S}}_{\mathtt{FMT}} = \Phi^{\mathtt{S}}_{\mathtt{FMF}}, \quad \Phi^{\mathtt{S}}_{\mathtt{TMT}} = \Phi^{\mathtt{S}}_{\mathtt{TMF}}, \quad \Phi^{\mathtt{S}}_{\mathtt{FRT}} = \Phi^{\mathtt{S}}_{\mathtt{TRT}}, \quad \Phi^{\mathtt{S}}_{\mathtt{FRF}} = \Phi^{\mathtt{S}}_{\mathtt{TRF}}$$
$$\Phi^{\mathtt{S}}_{\mathtt{TNT}} = \Phi^{\mathtt{S}}_{\mathtt{TNF}} = \Phi^{\mathtt{S}}_{\mathtt{FNT}} = \Phi^{\mathtt{S}}_{\mathtt{FNF}}$$

which can be used to simplify the stability analysis below.

While we introduced the dynamical system change associated with different events (i.e., with different combinations of sensor channel s, actuator channel a, and computational outcome c), so far we only described the deterministic evolution of the system. Provided that a specific set of events occurs during the evolution of the discrete part of our problem, the system behaviour is deterministic. However, the discrete state evolution of the closed-loop system is probabilistic and depends on the outcome of s, c, and a, which is here expressed via a Markov process.

3.3 Markov Chain

To take the discrete state evolution into account, we introduce a Markov chain. A Markov chain is a mathematical model for a stochastic process that describes a sequence of events or states, where the probability of transitioning from one state to another depends only on the current state and not on any previous states.

DEFINITION 4—MARKOV CHAIN
A Markov chain is defined by:

(i) A set V of N possible states, $V = \{v_1, v_2, \ldots, v_N\}$,

(ii) A transition probability matrix Π*, where the element* $\Pi_{i,j}$ *is the probability of transitioning from state* v_i *to state* v_j*, for all* $i, j \in \{1, 2, \ldots, N\}$,

(iii) The Markov property, which states that the probability of transitioning to a future state depends only on the current state and not on any past states.

The transition probability matrix Π *must satisfy the following two conditions:*

(i) $\Pi_{i,j} \geq 0, \forall i, j \in \{1, 2, ..., N\}$,

(ii) $\sum_{j=1}^{N} \Pi_{i,j} = 1, \forall i \in \{1, 2, ..., N\}$.

In our case, each state of the Markov chain represents an element of the set of outcomes, and directly maps to one of the matrices that govern the physical evolution of the system. The transition probabilities of the Markov chain depend on respectively (i) p_s – the probability of not receiving sensor data correctly, (ii) p_c – the probability of not completing the calculation of the control signal within one

period, and (iii) p_a – the probability of not receiving the actuator data correctly. As an example, the probability that in one period we transition to the state in which no faults occur, $s = \mathtt{T}$, $c = \mathtt{H}$, $a = \mathtt{T}$ is $(1-p_s)(1-p_c)(1-p_a)$. In such a discrete state, the discrete-time dynamics evolve according to $\Phi_{\mathtt{THT}}$. For compactness, we use 1_x to denote $(1-p_x)$, e.g., $1_s = (1-p_s)$.

Kill: For the `Kill` case, in principle there are 8 states in the Markov chain (stemming from the 8 possible matrices), but the equivalence between two pairs of matrices reduces the discrete states in which the system can be found to 6, corresponding to the closed-loop matrices $\Phi^{\mathtt{K}}_{\mathtt{THT}}$, $\Phi^{\mathtt{K}}_{\mathtt{FHT}}$, $\Phi^{\mathtt{K}}_{\mathtt{THF}}$, $\Phi^{\mathtt{K}}_{\mathtt{FHF}}$, $\Phi^{\mathtt{K}}_{\mathtt{FMX}}$ and $\Phi^{\mathtt{K}}_{\mathtt{TMX}}$, where `X` indicates that the outcome is irrelevant for this specific case. The Markov chain for the `Kill` strategy is encoded in the transition probability matrix

$$\Pi^{\mathtt{K}} = \begin{bmatrix} 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c \end{bmatrix}. \tag{9}$$

In $\Pi^{\mathtt{K}}$, the rows correspond to the dynamical system matrices: $\Phi^{\mathtt{K}}_{\mathtt{THT}}$, $\Phi^{\mathtt{K}}_{\mathtt{FHT}}$, $\Phi^{\mathtt{K}}_{\mathtt{THF}}$, $\Phi^{\mathtt{K}}_{\mathtt{FHF}}$, $\Phi^{\mathtt{K}}_{\mathtt{FMX}}$, $\Phi^{\mathtt{K}}_{\mathtt{TMX}}$. The transition matrix is fully connected, as from each state it is possible to reach any other state. Also, given the nature of the `Kill` action, every iteration of the control loop is independent. Therefore, the probability to reach any state in the Markov chain is the same, regardless of the current state, i.e., the rows in $\Pi^{\mathtt{K}}$ are identical.

Skip: In the `Skip` case, the transition matrix is not fully connected. In fact, each sequence of outcomes should satisfy Constraint 1, enforcing that `H` can only occur after either another `H` or `R`. Also, `N` must directly follow `M`, and a `R` can only follow a `M` or `N`. Hence, the transition matrix $\Pi^{\mathtt{S}}$ of the Markov chain for the continue and `Skip` case is

$$\Pi^{\mathtt{S}} = \begin{bmatrix} 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c & 0 & 0 & 0 \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c & 0 & 0 & 0 \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c & 0 & 0 & 0 \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_c & 1_c1_a & 1_cp_a \\ 0 & 0 & 0 & 0 & 0 & 0 & p_c & 1_c1_a & 1_cp_a \\ 0 & 0 & 0 & 0 & 0 & 0 & p_c & 1_c1_a & 1_cp_a \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c & 0 & 0 & 0 \\ 1_s1_c1_a & p_s1_c1_a & 1_s1_cp_a & p_s1_cp_a & p_sp_c & 1_sp_c & 0 & 0 & 0 \end{bmatrix}. \tag{10}$$

In $\Pi^{\mathtt{S}}$, the rows correspond to the following dynamical system matrices: $\Phi^{\mathtt{S}}_{\mathtt{THT}}$, $\Phi^{\mathtt{S}}_{\mathtt{FHT}}$, $\Phi^{\mathtt{S}}_{\mathtt{THF}}$, $\Phi^{\mathtt{S}}_{\mathtt{FHF}}$, $\Phi^{\mathtt{S}}_{\mathtt{FMX}}$, $\Phi^{\mathtt{S}}_{\mathtt{TMX}}$, $\Phi^{\mathtt{S}}_{\mathtt{XNX}}$, $\Phi^{\mathtt{S}}_{\mathtt{XRT}}$, $\Phi^{\mathtt{S}}_{\mathtt{XRF}}$. As can be seen for example in the fifth row, a miss can only be followed by either another miss or a recovery hit, i.e., a period in which no new job is released, but a previously released job is completed.

The transition probabilities are here treated as independent identically distributed (i.e., iid) random variables. Typically this is not the case in real systems. For example, if the sensor packet is lost, the probability of the control task overrunning its deadline likely increases. It is possible to cast said cases in our analysis framework using the theory of *conditional probabilities* [Dekking et al., 2006], i.e., probabilities that depend on the outcome of another event. This is done by considering vectors of p_s, p_c, and p_a in which each element represents the possibility of having a given number of faults in the corresponding event outcome, e.g., the second element of p_c represents the probability that the controller overruns in two consecutive periods. Then it is possible to create a Markov chain that handles every iteration starting from the corresponding state (e.g., the second element of p_c is only used when a first deadline overrun has occurred and the transition matrix has more zeros). Due to space limitations, we do not enter into details on this matter here. This is in particular interesting for p_c, as a deadline miss that follows another miss is less likely than the first deadline miss to occur when `Skip` is used.

3.4 Markov Jump Linear Systems Analysis

Discrete-Time Markov Jump Linear Systems [Costa et al., 2005] describe systems that can switch between different linear dynamics based on a finite set of discrete states. Informally, these systems combine the concepts of (discrete-time) switched linear systems and Markov chains. The system is subject to stochastic mode transitions described by a probability matrix (that specifies the likelihood of transitioning from one mode to another), i.e., the transition probability matrix of a Markov chain.

DEFINITION 5—DISCRETE-TIME AUTONOMOUS MARKOV JUMP LINEAR SYSTEM

A Discrete-Time Autonomous Markov Jump Linear System is a dynamical system (with initial state $\{\tilde{x}_0, \theta_0\}$),

$$\tilde{x}_{k+1} = \Phi_{\theta_k}\tilde{x}_k, \text{ where}$$

(i) Φ_{θ_k} *belongs to a set of discrete-time linear state-space models describing the evolution of the continuous dynamics, and*

(ii) $\{\theta_k\}_k$ *is a Markov process, governed by a Markov chain with transition probability matrix* Π.

In our case, the state-space models are given by the matrices derived in Section 3.2 and the Markov chain is the one described in Section 3.3.

The convergence of Markov Jump Linear Systems can be analysed using different tools, and in particular there are two main notions of stability: *mean stability* and *mean square stability*. Mean stability corresponds to convergence in probability,

while the second notion, mean square stability, corresponds to almost sure convergence.[4]

Mean stability is an important property for ensuring the robustness and reliability of stochastic systems, as it guarantees that the expected value of the system state will not exhibit unbounded growth over time. We analyse the expected value of the state $\tilde{x}_k$, i.e., $\mathbb{E}[\tilde{x}_k]$, and determine whether it converges to a specific value or not.

DEFINITION 6—MEAN STABILITY
The Discrete-Time Autonomous Markov Jump Linear System $\tilde{x}_{k+1} = \Phi_{\theta_k}\tilde{x}_k$ is mean stable if there exists a value μ such that for every initial state $\{\tilde{x}_0, \theta_0\}$, the expected value of the system state $\mathbb{E}[\tilde{x}_k] \to \mu$.

Mean square stability, on the other hand, analyses not only whether the expected value of the discrete-time system state converges, but also whether its covariance $\mathbb{E}\left[\tilde{x}_k\tilde{x}_k^{\mathrm{T}}\right]$ goes to zero; thus, implying *almost sure convergence*, i.e., both the probability of the system state converging and the probability of the state covariance going to zero goes to 1.

DEFINITION 7—MEAN SQUARE STABILITY
The Discrete-Time Autonomous Markov Jump Linear System $\tilde{x}_{k+1} = \Phi_{\theta_k}\tilde{x}_k$ is mean square stable if there exists a value μ such that for every initial state $\{\tilde{x}_0, \theta_0\}$,

$$\mathbb{E}[\tilde{x}_k] \to \mu, \quad \mathbb{E}\left[\tilde{x}_k\tilde{x}_k^{\mathrm{T}}\right] \to 0.$$

Mean square stability is a desirable property for stochastic systems because it implies mean stability, but also provides more information about the rate of decay of the system's fluctuations. In particular, it implies that the fluctuations experienced by the system will decay exponentially fast over time. Mean square stability is closely related to the notion of Shur stability for deterministic systems presented in Equation (5), and is commonly used in the analysis and design of stochastic control and estimation algorithms.

We want to test that the systems subject to fault are mean square stable. As extensively discussed in the literature [Costa et al., 2005], testing for mean square stability implies calculating the eigenvalues of the operator Ψ representing the evolution of the Markov Jump Linear System's covariance matrix. In particular, a Discrete-Time Autonomous Markov Jump Linear System is mean square stable if and only if

$$\begin{aligned} \rho(\Psi) &= \max|\operatorname{eig}(\Psi)| < 1 \\ \Psi &= (\Pi^{\mathrm{T}} \otimes I_{n^2}) \cdot \operatorname{blkdiag}\left(\{\Phi_i^{\mathrm{T}} \otimes \Phi_i\}_i\right). \end{aligned} \tag{11}$$

Here, $\otimes$ represents the Kronecker product. The matrix Π is one of the Markov chain transition matrix specified either in Equation (9) or in Equation (10) depending on

[4] Other notions of stability also exist, like stochastic stability and mean square exponential stability. However, if a system is mean square stable it is also mean stochastically stable as well as mean square exponentially stable.

the deadline overrun strategy adopted. Furthermore, I_{n^2} is the identity matrix of size n^2, where $n = n_x + n_z + n_u + n_y$ is the order of the closed-loop matrices Φ_i (that corresponds to the actual values of the matrices Φ_{θ_k} from Defintion 5, and hence to the matrices specified either in Equation (7) or in Equation (8) depending on the deadline overrun strategy adopted). Finally, the last term is a block diagonal Kronecker product of the matrices in the possible closed-loop system realisations.

We analyse the mean square stability of the system by constructing the operator Ψ and calculating its eigenvalues, and hence $\rho(\Psi)$. The calculation of $\rho(\Psi)$ is not demanding for small- and medium-scale systems. The time-consuming part of analysing the mean square stability comes from the eigenvalue decomposition.[5] We emphasise that the Markov Jump Linear Systems covariance matrix Ψ is sparse, implying that a speedup could be achieved if this sparsity is taken into account, for instance by utilising the Lanczsos algorithm for computing the largest magnitude eigenvalues [Golub and Loan, 1996].

4. Evaluation

In this section we apply the Markov Jump Linear Systems stability analysis presented in Section 3 to two case studies lifted from the literature on controllers that experience faults[6]:

- In Section 4.1 we apply our analysis to an automotive cruise control system controlled with a state-feedback controller, taken from [Ghosh et al., 2018];
- In Section 4.2, we analyse a ball and beam process controlled by a Linear-Quadratic-Gaussian (LQG) controller, taken from [Vreman et al., 2022].

We assume that both computational overruns and IO channel packet losses are Bernoulli distributed [Schenato et al., 2007], i.e., that the outcomes s, c, and a are independent and identically distributed random variables. We denote with p_s the probability of losing a packet on the sensor channel; with p_c the probability of the control task overrunning a deadline; and with p_a the probability of losing a packet on the actuator channel.

For each case study, we perform the following three sets of experiments:

(i) We fix *two* of the probabilities p_s, p_c, and p_a to 0 and vary the remaining one between 0 and 1 (excluded) with a step of 0.01, i.e., $p_x \in \{0, 0.01, \ldots 0.99\}$, $x \in \{s, c, a\}$.

[5] The computational complexity of eigenvalue decomposition using the Coppersmith and Winograd algorithm is $O(n^{2.376})$. For mean square stability, we compute the eigenvalues of a matrix with dimension $n^2 m$, where m is the dimension of transition matrix Π. The complexity is thus $O(n^{4.752} m^{2.376}) \approx O(n^{4.752})$ (since m is a constant). Based on empirical tests, for eight-dimensional systems (i.e., systems in which $n = 8$) the analysis takes at most 1 second.

[6] A third case study was investigated, but it was excluded due to space limitations.

(ii) We fix *one* of the probabilities p_s, p_c, and p_a to 0 and vary the remaining two between 0 and 1 (excluded) with a step of 0.01.

(iii) We analyse the closed-loop system dynamics when the sensor channel experiences 15% traffic (packet) loss, the controller executes in a busy real-time operating system and thus overruns 40% of its deadlines, and the actuator channel has a probability of losing 5% of its packets. This corresponds to $p_s = 0.15$, $p_c = 0.4$, and $p_a = 0.05$.

We analyse controllers that are implemented with `Kill` and `Skip` as deadline overrun handling strategy and `Zero` and `Hold` as actuation modes. For all experiments, we calculate $\rho(\Psi)$ to determine the closed-loop mean square stability according to Equation (11).

4.1 Automotive Cruise Control Evaluation

Ghosh et al. [Ghosh et al., 2018] present a method to derive a fault-tolerant state-feedback controller to address stochastic computational faults. Additionally, the performance and stability of said controller are validated on the model of an automotive cruise control system, which we denote with $\mathcal{P}_1$. The plant is inherently stable, i.e., $\rho(A) < 1$. In this paper, we analyse the automotive cruise control system controlled by a baseline state-feedback controller $\mathcal{C}_1$, also presented in [Ghosh et al., 2018].

$$\mathcal{P}_1 : \begin{cases} x_{k+1} & = A x_k + B u_k \\ y_k & = x_k \end{cases}, \quad \mathcal{C}_1 : u_{k+1} = K x_k,$$

$$A = \begin{bmatrix} 1 & 0.01 & 0 \\ -0.0003 & 0.9997 & 0.01 \\ -0.0604 & -0.0531 & 0.9974 \end{bmatrix}, B = \begin{bmatrix} 0.0001 \\ 0.0001 \\ 0.0247 \end{bmatrix}$$

$$K = \begin{bmatrix} -872.54 & -131.49 & -10.097 \end{bmatrix}$$

Experiment (i) Figure 2 shows the results of experiment (i). Each plot corresponds to a particular strategy combination (e.g., `Kill&Zero`) and the x-axis shows the probability that is varied, $p_x \in \{0, 0.01, \ldots, 0.99\}$, while the y-axis shows the result of the analysis, $\rho(\Psi)$. A value of $\rho(\Psi)$ that exceeds 1 implies that the system with the given p_x is not mean square stable.

When the sensor channel's packet loss probability p_s increases, so does the magnitude of the closed-loop system's eigenvalues, no matter the choice of strategy to handle the deadline miss and the actuation mode. In the case of the sensor packet loss plot, $p_s \neq 0$ and $p_c = p_a = 0$. Hence, the controller will never miss a deadline and the actuator will always output a new control signal, implying that losing packets on the sensor channel are invariant to the choice of deadline overrun strategy and actuation mode.

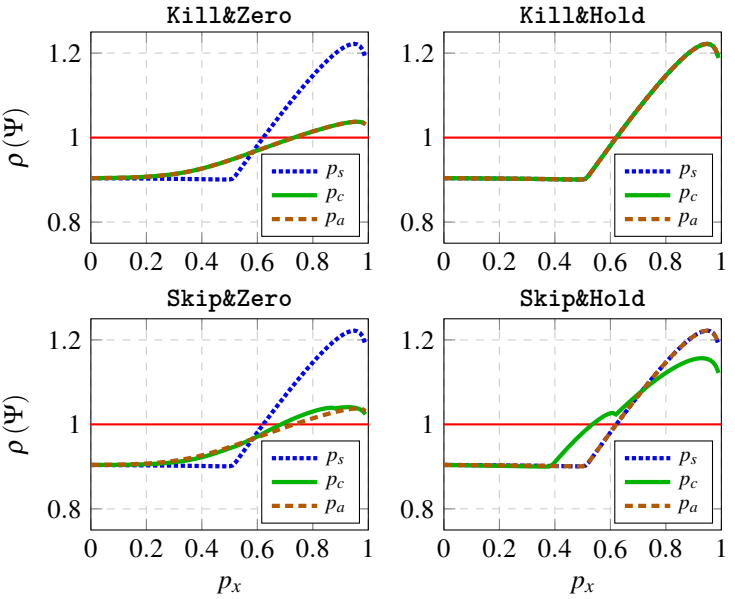


Figure 2. Results of Experiment (i) on the cruise control plant [Ghosh et al., 2018].

When $p_s = p_c = 0$ and $p_a \neq 0$, the stability of the system depends on the choice of actuation mode, but is agnostic to the deadline overrun strategy. When the `Hold` actuation mode is favoured, the actuator outcome is comparable to missing a sensor packet and continuing the controller execution, since the sensor packets always arrive ($p_s = 0$) and the controller never overruns its deadline ($p_c = 0$). Additionally, we deduce that the automotive cruise control system tolerates a higher probability of losing actuator commands before going unstable under the `Zero` actuation mode than under `Hold`. This likely follows from the system being inherently stable.

For the `Kill` overrun strategy, the eigenvalue of Ψ with the largest magnitude is identical for computational overruns and lost actuator packets. Since the controller is a static state-feedback law, as soon as a deadline is hit a new control signal is computed without any residual problems originating from a diverged control state (state-feedback controllers are stateless). However, when computational overruns occur and the scheduler has adopted the `Skip` strategy, the computed control signal is based on old plant states, thus negatively impacting the robustness of the system.

The shape of the curve representing the computational overruns, i.e., the case when $p_c \neq 0$, $p_s = p_a = 0$, is strongly dependent on the system dynamics. When the probability of overrunning a deadline goes to 1, the system runs in *open-loop*, i.e., without any feedback. Since the plant is stable, running in open-loop system would preserve the stability of the system, with $\rho(\Psi)$ being close to 1. In fact, the *switch-*

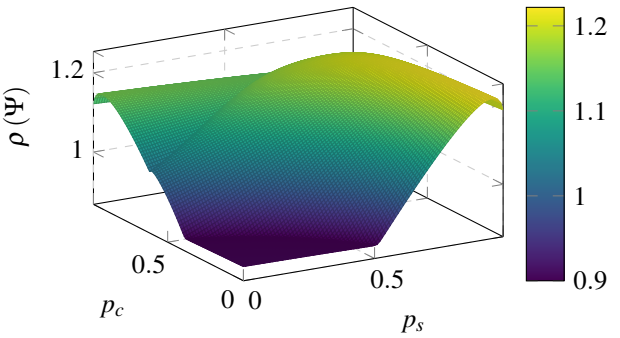


Figure 3. Results of Experiment (ii) on the cruise control plant [Ghosh et al., 2018] for (continue) `Skip&Hold`.

ing between meeting and overrunning deadlines is the cause of destabilisation in the closed-loop system. This is consistent with observations presented in [Vreman et al., 2021] about how the closed-loop robustness can improve with an increased number of computational overruns. We do however emphasise that despite the value of $\rho(\Psi)$ decreasing, it is still above 1, thus indicating that the system is *not* mean square stable.

Experiment (ii) Figure 3 displays a 3d plot of $\rho(\Psi)$ when both sensor packet losses and computational overruns may occur, but the actuator packets are always delivered correctly, i.e., $p_s \neq 0$, $p_c \neq 0$, and $p_a = 0$, for `Skip&Hold`.[7] The x- and y-axes show the probabilities p_s and p_c, while the z-axes corresponds to $\rho(\Psi)$. It is possible to recognise the curves of $p_s = 0$ and $p_c = 0$ from Figure 2. For configurations where both sensor channel losses and computational overruns are present, the system robustness is degraded. As an example, individually when $p_s = 0.51$ or $p_c = 0.51$ the system is stable (see Figure 2), but when both values are set to 0.51, the system is unstable, as can be seen in Figure 3.

Experiment (iii) Table 1 shows the results of Experiment (iii).

Table 1. Results of Experiment (iii) on the automotive cruise control.

	`Kill&Zero`	`Kill&Hold`	`Skip&Zero`	`Skip&Hold`
$\rho(\Psi)$	0.9313	0.9006	0.9274	0.9638

Regardless of deadline overrun strategy and actuation mode, the Markov Jump Linear System is stable when $p_s = 0.15$, $p_c = 0.4$, and $p_a = 0.05$. The results confirm that the cruise control system is robust to simultaneous occurrences of multiple

[7] Experiments with different configurations or strategies do not provide additional insights and are hence not reported due to space limitations.

fault types. It is interesting to note that the outcome configuration $(p_s, p_c, p_a) = (0, 0.4, 0)$ leads to $\rho(\Psi) = 0.9108$ for the `Skip&Hold` strategy. In other words, despite the faults on the IO channels appearing to be inconsequential for the system stability (for $p_s < 0.5$ and $p_a < 0.5$) in Figure 2, they significantly affect the dynamics of the system. In fact, perturbing the probabilities on the IO channels from Experiment (iii) by 8% to $(p_s, p_c, p_a) = (0.23, 0.4, 0.13)$, the system is unstable with a value of $\rho(\Psi) = 1.0014$.

4.2 Ball and Beam Evaluation

Vreman et al. [Vreman et al., 2022] propose a controller implementation method that aims at improving the performance of systems where the controller is subject to probabilistic deadline overruns. The implementation method is evaluated on a physical ball and beam plant, $\mathcal{P}_2$, controlled using a linear-quadratic-Gaussian (LQG) controller $\mathcal{C}_2$.

$$\mathcal{P}_2 : \begin{cases} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{cases}, \quad \mathcal{C}_2 : \begin{cases} z_{k+1} &= Fz_k + Gy_k \\ u_{k+1} &= Hz_k + Ky_k \end{cases}$$

$$A = \begin{bmatrix} 1 & 0 & 0 \\ -0.1 & 1 & 0 \\ -0.0005 & 0.01 & 1 \end{bmatrix}, B = \begin{bmatrix} 0.045 \\ -0.0023 \\ -7.5 \cdot 10^{-6} \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$F = \begin{bmatrix} 0.709 & 0.054 & 0.041 \\ 0.011 & 0.997 & -0.219 \\ 0.004 & 0.010 & 0.934 \end{bmatrix}, G = \begin{bmatrix} 0.152 & 0.001 \\ -0.104 & 0.217 \\ -0.004 & 0.066 \end{bmatrix},$$

$$H = \begin{bmatrix} -2.433 & 1.201 & 0.562 \end{bmatrix}, K = \begin{bmatrix} -0.672 & 0.368 \end{bmatrix}.$$

Note that the ball and beam plant is modelled as a triple integrator, i.e., there are three eigenvalues with magnitude $\rho(A) = 1$, making the system unstable.

Experiment (i) Figure 4 shows the results of Experiment (i). Unlike the cruise control system, there now exists a clear distinction between the three different cases of Experiment (i). It is evident that the sensor, actuator, and controller all affect the Markov Jump Linear System stability individually, and can thus not be seen as equivalent outcomes.

The contrast between the behaviour of the ball and beam system and the prior setup come from the controller dynamics in $\mathcal{C}_2$. While $\mathcal{C}_1$ is stateless, the LQG controller $\mathcal{C}_2$ does have an internal state, meaning that if the controller overruns a deadline or an IO channel packet is lost, it will also impact the system negatively for some time after the event. Despite the introduced controller dynamics, the closed-loop ball and beam system appear to be robust to both IO channel packet losses (up to p_s and p_a around 0.85) and computational overruns (up to p_c around 0.35 for the `Zero` actuation mode and 0.5 for `Hold` actuation mode).

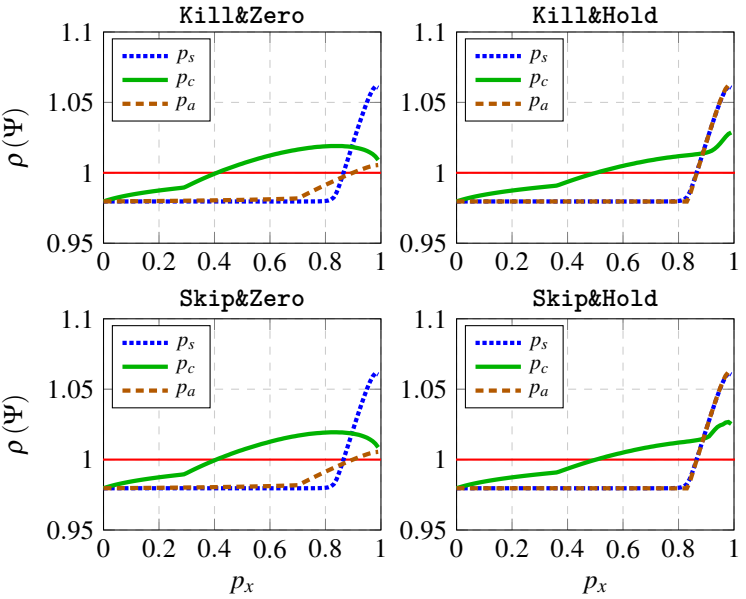


Figure 4. Results of Experiment (i) on the ball and beam plant [Vreman et al., 2022].

Experiment (ii) The surface plot in Figure 5 shows $\rho(\Psi)$ with varying p_s and p_a, assuming that the controller always succeeds in meeting its computational deadline ($p_c = 0$) and that the `Zero` actuation mode is adopted. It is easy to see that even for high values of both p_s and p_a, the system remains mean square stable as long as the controller does not miss its deadlines. This is likely a result of the controller being designed with robustness as a design objective.

Experiment (iii) Table 2 presents the results acquired from Experiment (iii) on the ball and beam system.

Table 2. Results of Experiment (iii) on the ball and beam example.

	`Kill&Zero`	`Kill&Hold`	`Skip&Zero`	`Skip&Hold`
$\rho(\Psi)$	1.0000	0.9936	1.0002	0.9936

The results indicate that the choice of actuation mode is affecting $\rho(\Psi)$ more than the choice of deadline overrun strategy. In fact, both implementations employing the `Zero` actuation mode are unstable, while if the `Hold` actuation mode is selected, the closed-loop system is mean square stable under the tested probabilities. When we increase the probability of the IO channels experiencing packet losses,

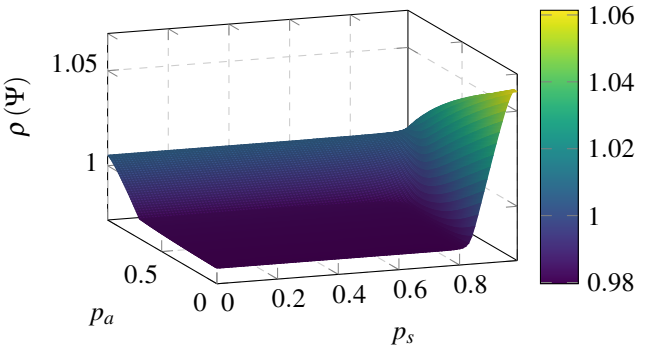


Figure 5. Results of Experiment (ii) on the ball and beam [Vreman et al., 2022] for (continue) `Kill&Zero` as deadline handling strategy and actuation mode.

the `Zero` actuation mode quickly becomes unstable, while the `Hold` actuation mode can tolerate up to 75% packet loss on *both* the sensor and actuator channels simultaneously before becoming mean square unstable, i.e., before $\rho(\Psi) \geq 1$. This likely follows from the model of the system being a triple integrator, where zeroing the output signal in case of a packet loss or computational overrun drives the system state away from the desired value. Instead, holding the previous control command keeps the integrator state close to its ideal value.

5. Related Work

In recent years, probabilistic analysis techniques for real-time systems are becoming more prominent [Davis and Cucu-Grosjean, 2019]. In particular, a few interesting methods have been developed to compute the probability that specific tasks miss their deadlines, e.g., [Brüggen et al., 2021]. In [Brüggen et al., 2018], the authors propose a method to safely estimate the deadline miss rate of tasks in a uniprocessor system under a preemptive fixed-priority scheduling policy. For mixed-criticality systems, [Maxim et al., 2017] introduce a probabilistic analysis method for analysing worst-case execution time distributions, and in turn worst-case deadline miss probabilities, under fixed-priority preemptive scheduling. An efficient and accurate convolution-based approach to calculating deadline miss probabilities is introduced in [Chen et al., 2018]. The authors of [Markovi et al., 2021] propose a method for down-sampling the random variables in order to improve space and time complexity of the analysis methods.

The ratios provided by the deadline miss probability analysis can be utilised to improve the design of control systems. In [Schenato et al., 2007], the authors derive an optimal controller (control law and estimator) for a networked control system where packet arrivals are modelled stochastically. The authors of [Cloosterman et

al., 2010] propose a stabilising controller for networked control systems subject to stochastic network delays and packet losses. Pazzaglia et al. [Pazzaglia et al., 2019] derive a robust controller to be used when the control task can experience deadline misses stochastically.

Generally, fault-tolerant controllers are designed to counteract one specific type of fault, e.g., sensor losses *or* deadline overruns. To analyse whether a system is robust to the specific type of fault or not, different analysis methods have been proposed. A stability analysis method for control systems subject to weakly-hard packet losses is proposed in [Linsenmayer et al., 2020]. In [Maggio et al., 2020], the authors propose a method for analysing the stability of real-time control systems where the control task is subject to consecutive deadline overruns.

6. Conclusion and Future Work

There exists a common misconception that analysing a control system's robustness to packet losses on the network is equivalent to having the control algorithm overrun its timing budget, and vice versa. This paper proposes an approach to analyse real-time control systems and their stability properties when subject to multiple types of faults. In particular, we analyse *simultaneous* packet losses on the IO communication channels and computational overruns of the task executing the control algorithm, making the analysis more comprehensive than the state-of-the-art alternatives.

We envision that the analysis method and the corresponding experimental campaign will be used to improve future analysis methods and correct any misconceptions about how faults interact in computer-controlled systems. Finally, the paper brings the control analysis closer to the state-of-practice compared to the research literature, because it relies on a probabilistic failure model. In industrial setups, it is in fact easier to get estimates of the probability of certain events from testing campaigns, rather than to extract complex (but deterministic) guarantees like the validity of a weakly-hard constraint.

Acknowledgements

This work was supported by the ELLIIT Strategic Research Area. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project). This (publication/report) reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

References

Ahrendts, L., S. Quinton, T. Boroske, and R. Ernst (2018). "Verifying weakly-hard real-time properties of traffic streams in switched networks". In: *30th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 15:1–15:22. ISBN: 978-3-95977-075-0.

Åström, K. J. and B. Wittenmark (1997). *Computer-Controlled Systems (3rd Ed.)* Prentice-Hall, Inc., USA. ISBN: 0133148998.

Åström, K. (1970). *Introduction to stochastic control theory*. Vol. 70. Mathematics in science and engineering. Academic Press, United States. ISBN: 0-12-065650-7.

Bernat, G., A. Burns, and A. Liamosi (2001). "Weakly hard real-time systems". *IEEE Transactions on Computers* **50**:4, pp. 308–321. DOI: 10.1109/12.919277.

Blair Jr., W. P. and D. D. Sworder (1975). "Feedback control of a class of linear discrete systems with jump parameters and quadratic cost criteria". *International Journal of Control* **21**:5, pp. 833–841. DOI: 10.1080/00207177508922037.

Bohrer, B., Y. K. Tan, S. Mitsch, M. O. Myreen, and A. Platzer (2018). "Veriphy: verified controller executables from verified cyber-physical system models". In: *39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2018. Philadelphia, PA, USA, pp. 617–630. ISBN: 9781450356985. DOI: 10.1145/3192366.3192406.

Bolzern, P., P. Colaneri, and G. De Nicolao (2010). "Markov jump linear systems with switching transition rates: mean square stability with dwell-time". *Automatica* **46**:6, pp. 1081–1088. ISSN: 0005-1098. DOI: doi.org/10.1016/j.automatica.2010.03.007.

Brüggen, G. von der, N. Piatkowski, K.-H. Chen, J.-J. Chen, and K. Morik (2018). "Efficiently approximating the probability of deadline misses in real-time systems". In: *30th Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-075-0.

Brüggen, G. von der, N. Piatkowski, K. Chen, J. Chen, K. Morik, and B. B. Brandenburg (2021). "Efficiently approximating the worst-case deadline failure probability under EDF". In: *42nd IEEE Real-Time Systems Symposium, RTSS*. DOI: 10.1109/RTSS52674.2021.00029. URL: doi.org/10.1109/RTSS52674.2021.00029.

Buttazzo, G., G. Lipari, L. Abeni, and M. Caccamo (2005). *Soft Real-Time Systems*. Springer.

Cervin, A. (2005). "Analysis of overrun strategies in periodic control tasks". *IFAC Proceedings Volumes* **38**:1. 16th IFAC World Congress, pp. 219–224. ISSN: 1474-6670. DOI: 10.3182/20050703-6-CZ-1902.01076.

Chen, K., G. Von Der Brüggen, and J. Chen (2018). "Analysis of deadline miss rates for uniprocessor fixed-priority scheduling". In: *24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 168–178.

Chen, K., N. Ueter, G. von der Brüggen, and J.-J. Chen (2019). "Efficient computation of deadline-miss probability and potential pitfalls". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 896–901. DOI: 10.23919/DATE.2019.8714908.

Cloosterman, M. B. G., L. Hetel, N. van de Wouw, W. P. M. H. Heemels, J. Daafouz, and H. Nijmeijer (2010). "Controller synthesis for networked control systems". *Automatica* **46**:10, pp. 1584–1594. ISSN: 0005-1098. DOI: doi.org/10.1016/j.automatica.2010.06.017.

Costa, O. L. V., R. P. Marques, and M. D. Fragoso (2005). *Discrete-Time Markov Jump Linear Systems*. Probability and Its Applications. Springer London. ISBN: 978-1-85233-761-2.

Davis, R. I. and L. Cucu-Grosjean (2019). "A survey of probabilistic timing analysis techniques for real-time systems". *Leibniz Transactions on Embedded Systems* **6**:1. DOI: 10.4230/LITES-v006-i001-a003.

Dekking, F. M., C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester (2006). *A Modern Introduction to Probability and Statistics*. ISBN: 978-1-84628-168-6.

Donkers, M. C. F., W. P. M. H. Heemels, N. van de Wouw, and L. Hetel (2011). "Stability analysis of networked control systems using a switched linear systems approach". *IEEE Transactions on Automatic Control* **56**:9, pp. 2101–2115. DOI: 10.1109/TAC.2011.2107631.

Fang, Y. and K. Loparo (2002). "Stochastic stability of jump linear systems". *IEEE Transactions on Automatic Control* **47**:7, pp. 1204–1208. DOI: 10.1109/TAC.2002.800674.

Ghosh, S. K., S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty (2018). "Design and validation of fault-tolerant embedded controllers". In: *Design, Automation & Test in Europe Conference*, pp. 1283–1288. DOI: 10.23919/DATE.2018.8342212. URL: doi.org/10.23919/DATE.2018.8342212.

Golub, G. H. and C. F. van Loan (1996). *Matrix Computations (3rd Ed.)* USA. ISBN: 0801854148.

Goswami, D., D. Mueller-Gritschneder, T. Basten, U. Schlichtmann, and S. Chakraborty (2014). "Fault-tolerant embedded control systems for unreliable hardware". In: *International Symposium on Integrated Circuits*.

Henzinger, T., B. Horowitz, and C. Kirsch (2003). "Giotto: A time-triggered language for embedded programming". *Proceedings of the IEEE* **91**:1, pp. 84–99. DOI: 10.1109/JPROC.2002.805825.

Hobbs, C., B. Ghosh, S. Xu, P. S. Duggirala, and S. Chakraborty (2022). "Safety analysis of embedded controllers under implementation platform timing uncertainties". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **41**:11, pp. 4016–4027. DOI: 10.1109/TCAD.2022.3198905.

Horssen, E. P. van, A. R. B. Behrouzian, D. Goswami, D. Antunes, T. Basten, and W. P. M. H. Heemels (2016). "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses". In: *2016 European Control Conference (ECC)*, pp. 2571–2577. DOI: 10.1109/ECC.2016.7810677.

Kauer, M., D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. M. Annaswamy (2014). "Fault-tolerant control synthesis and verification of distributed embedded systems". In: *Design, Automation & Test in Europe Conference Exhibition*.

Liberzon, D. (2014). "Finite data-rate feedback stabilization of switched and hybrid linear systems". *Automatica* **50**:2, pp. 409–420. ISSN: 0005-1098. DOI: doi.org/10.1016/j.automatica.2013.11.037.

Lincoln, B. and A. Cervin (2002). "Jitterbug: a tool for analysis of real-time control performance". In: *IEEE Conference on Decision and Control*. Vol. 2, pp. 1319–1324. DOI: 10.1109/CDC.2002.1184698.

Ling, Q. and M. D. Lemmon (2002). "Robust performance of soft real-time networked control systems with data dropouts". In: *41st IEEE Conference on Decision and Control*. Vol. 2. DOI: 10.1109/CDC.2002.1184681.

Linsenmayer, S. and F. Allgower (2017). "Stabilization of networked control systems with weakly hard real-time dropout description". In: *56th IEEE Conference on Decision and Control (CDC)*, pp. 4765–4770.

Linsenmayer, S., M. Hertneck, and F. Allgower (2020). "Linear weakly hard real-time control systems: time- and event-triggered stabilization". *IEEE Transactions on Automatic Control*.

Ma, Y., Y. Wang, S. Cairano, T. Koike-Akino, J. Guo, P. Orlik, and C. Lu (2018). "A smart actuation architecture for wireless networked control systems". In: DOI: 10.1109/CDC.2018.8619831.

Maggio, M., A. Hamann, E. Mayer-John, and D. Ziegenbein (2020). "Control-system stability under consecutive deadline misses constraints". In: *32nd Euromicro Conference on Real-Time Systems (ECRTS)*. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Markovi, F., A. V. Papadopoulos, and T. Nolte (2021). "On the Convolution Efficiency for Probabilistic Analysis of Real-Time Systems". In: *33rd Euromicro Conference on Real-Time Systems*. DOI: 10.4230/LIPIcs.ECRTS.2021.16.

Maxim, D., R. I. Davis, L. Cucu-Grosjean, and A. Easwaran (2017). "Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling". In: *25th International Conference on Real-Time Networks and Systems*. DOI: 10.1145/3139258.3139276.

Ohlin, M., D. Henriksson, and A. Cervin (2006). *Truetime 1.4 - reference manual*.

Pazzaglia, P., C. Mandrioli, M. Maggio, and A. Cervin (2019). "DMAC: Deadline-Miss-Aware Control". In: *31st Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 133. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 1:1–1:24. ISBN: 978-3-95977-110-8.

Pazzaglia, P., L. Pannocchi, A. Biondi, and M. D. Natale (2018). "Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses". In: Altmeyer, S. (Ed.). *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:22. ISBN: 978-3-95977-075-0. DOI: 10.4230/LIPIcs.ECRTS.2018.10.

Safari, S., M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, and J. Henkel (2022). "A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues". *IEEE Access* **10**, pp. 12229–12251.

Schenato, L. (2009). "To zero or to hold control inputs with lossy links?" *IEEE Transactions on Automatic Control* **54**:5, pp. 1093–1099.

Schenato, L., B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry (2007). "Foundations of control and estimation over lossy networks". *Proceedings of the IEEE* **95**:1, pp. 163–187. DOI: 10.1109/JPROC.2006.887306.

Schinkel, M., W.-H. C., and A. Rantzer (2002). "Optimal control for systems with varying sampling rate". In: *2002 American Control Conference*. Vol. 4, 2979–2984 vol.4. DOI: 10.1109/ACC.2002.1025245.

Stankovic, J., M. Spuri, M. D. Natale, and G. Buttazzo (1995). "Implications of classical scheduling results for real-time systems". *Computer* **28**:6, pp. 16–25. DOI: 10.1109/2.386982.

Sun, Y. and M. D. Natale (2017). "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks". *ACM Transactions on Embedded Computing Systems* **16**:5s. ISSN: 1539-9087.

Vreman, N., A. Cervin, and M. Maggio (2021). "Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses". In: *33rd Euromicro Conference on Real-Time Systems (ECRTS)*. Vol. 196. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN: 978-3-95977-192-4. DOI: 10.4230/LIPIcs.ECRTS.2021.15.

Vreman, N., R. Pates, and M. Maggio (2022). "Weaklyhard.jl: scalable analysis of weakly-hard constraints". In: *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 228–240. DOI: 10.1109/RTAS54340.2022.00026.

Wang, W., P. Mishra, and A. Gordon-Ross (2012). "Dynamic cache reconfiguration for soft real-time systems". *ACM Trans. Embed. Comput. Syst.* **11**:2. ISSN: 1539-9087. DOI: 10.1145/2220336.2220340.

Yang, L. and N. Ozay (2021). "Safety control synthesis for systems with missing measurements". In: *IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021*. Vol. 54. 5, pp. 97–102. DOI: doi.org/10.1016/j.ifacol.2021.08.481.

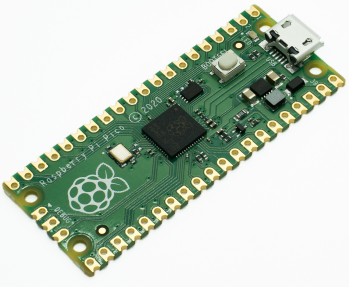
Vad är det värsta som kan hända när datorn krånglar?

Nils Vreman
Institutionen för Reglerteknik

Populärvetenskaplig sammanfattning av doktorsavhandlingen *Analysis of Embedded Controllers Subject to Computational Overruns*, juni 2023. Avhandlingen kan laddas ner från: http://www.control.lth.se/publications

Datorer i olika storlekar och med olika styrka finns runt omkring oss hela tiden. Allt från mobilen i din ficka till bilen du åker med till jobbet innehåller många små datorer, också kallade *inbyggda system*. Varje litet inbyggt system har uppgifter som måste uträttas, t.ex., beräkna hastigheten. I de flesta fall finns det krav och specifikationer på att beräkningarna måste uträttas inom en viss tidsram. Till exempel, om bromsen trycks in förväntas bilen sakta ner direkt. I verkligheten tar det lite tid för det inbyggda systemet att beräkna nedbromsningshastigheten innan kommandot kan skickas till bromssystemet. Systemet är dock alltid designat med hänsyn till den extra tiden det tar att uträtta hela uppgiften.

Tyvärr kan inbyggda system krångla, precis som en persondator. Bland annat kan bakgrundsprogram, virus, och föråldrad hårdvara/programvara störa beräkningen. Det kan försena beräkningen, och i värsta fall medföra att uppgiften inte blir klar alls. Vad händer då om det inbyggda systemet krånglar på ett sådant sätt att tidsramen för beräkningen inte respekteras? Just den frågan besvaras i den här avhandlingen.



Exempel på ett inbyggt system. På bilden ses en Raspberry Pi Pico där det svarta chippet i mitten (storlek 7×7 mm) är en mikrokontroller, dvs., ett inbyggt system.[1]

Många inbyggda system sitter i apparater där det inte gör så mycket om något krånglar. Till exempel har mycket hemelektronik (så som moderna stereosystem, TV-apparater, och kylskåp) inbyggda system för att styra funktionaliteten och kommunicera med andra smarta system. Det är irriterande om smart-TVns bild blir oskarp, men det är inget som påverkar oss märkbart. Värre konsekvenser uppstår om det inbyggda systemet krånglar i bilens bromssystem, en pacemaker, eller i ett

[1] Laserlicht, CC BY-SA 4.0, https://creativecommons.org/licenses/by-sa/4.0/, via Wikimedia Commons

flygplans styrsystem. Då kan det uppstå verklig fara för människor i närheten och värstascenariot inkluderar även dödsfall.

Den här avhandlingen utvecklar matematiska verktyg och metoder för att förbättra analysen av dessa säkerhetskritiska inbyggda system. Det är livsviktigt att ingen person kommer till skada och därför fokuserar analysen på att se till att hela systemet är säkert, även under väldigt dåliga förhållanden. Dåliga förhållanden innebär här, bland annat, att det inbyggda systemet inte lyckas beräkna något under en väldigt lång tid på grund av, t.ex., en hacker-attack. Med förhandsinformation om hur länge systemet är säkert när något går fel är det möjligt att utveckla metoder för att säkert stänga av eller starta om systemet innan någon kommer till skada.

Ett säkerhetskritiskt system blir inte nödvändigtvis osäkert bara för att beräkningen tar lite extra tid då och då, men det kan göra att upplevelsen blir väldigt obehaglig för användaren. Till exempel kan bilens nedbromsning bli ryckig och oregelbunden. På lång sikt är det då troligt att systemet bryts ner och går sönder på grund av utförandets oregelbundenhet, och som konsument överger du antagligen tillverkaren. Detta kan kosta tillverkaren väldigt mycket pengar och försämrar generellt sett företagets rykte. Därför räcker det inte att enbart se till att produkten är helt säker, den måste också nå upp till en viss kvalitet för att det ska bli lönsamt att producera den.

I avhandlingen utformas både algoritmer för att förbättra produktkvaliteten och metoder för att kvalitetsgranska slutprodukten. Genom att förändra beräkningarna som utförs av det inbyggda systemet i realtid visas att det är möjligt att kompensera för den försämrade prestandan. Nya metoder för att undersöka prestandan utvecklas och används för att garantera att algoritmerna fungerar enligt förväntan när det inbyggda systemet krånglar. För att bekräfta att avhandlingens bidrag går att använda i verkligheten testas både algoritmerna, säkerhetsanalysen och prestandaanalysen på flertalet verkliga system. Experimenten visar att metoderna både är enkla att använda och ger en bra bild av hur det verkliga systemet påverkas av problem med de inbyggda systemen.