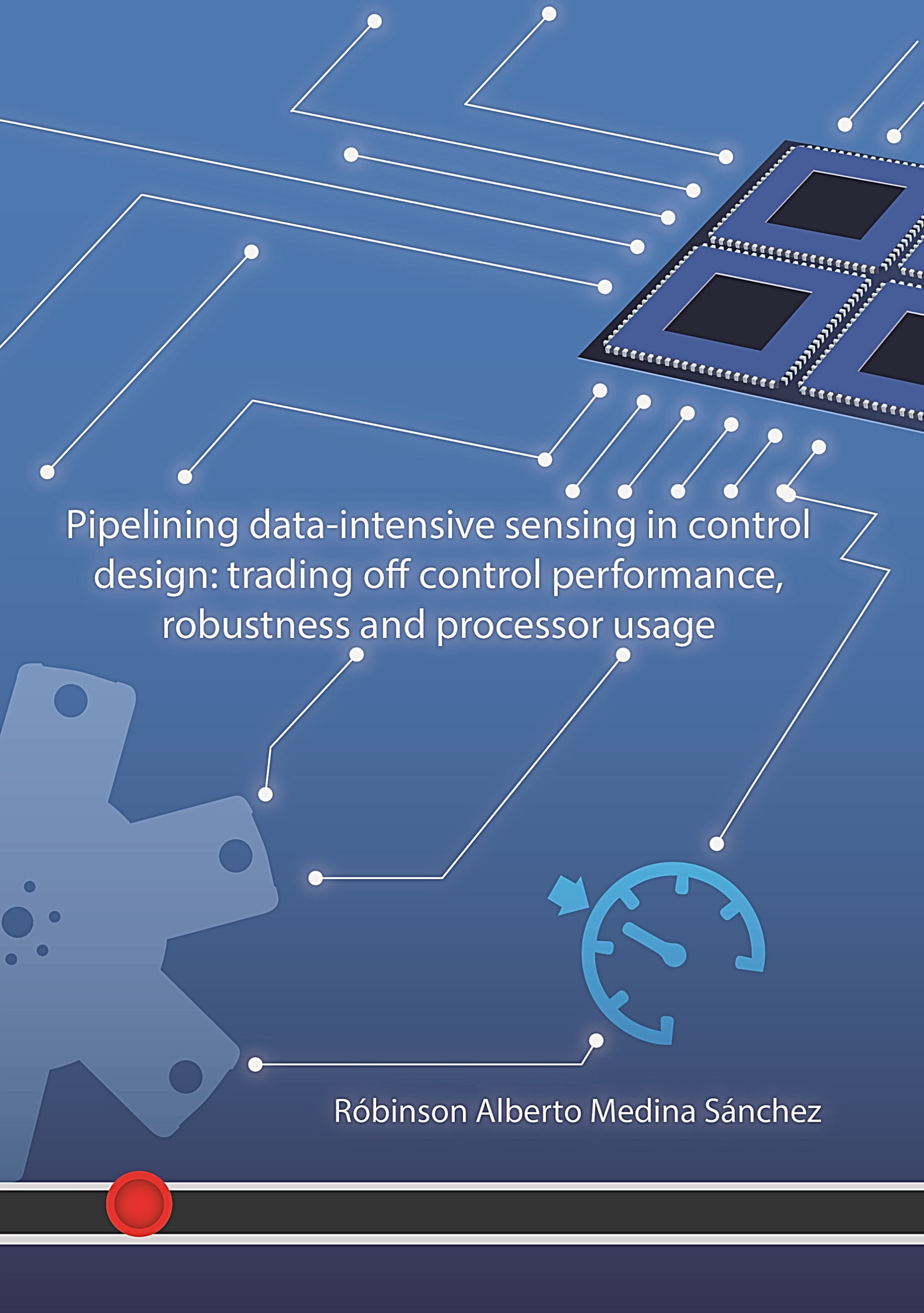
Pipelining data-intensive sensing in control design



Pipelining data-intensive sensing in control design: trading off control performance, robustness and processor usage

Róbinson Alberto Medina Sánchez

Pipelining data-intensive sensing in control design: trading off control performance, robustness and processor usage

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op vrijdag 22 oktober 2021 om 16:00 uur.

door

Róbinson Alberto Medina Sánchez

geboren te Ibagué, Tolima, Colombia

Pipelining data-intensive sensing in control design: trading off control performance, robustness and processor usage

Róbinson Alberto Medina Sánchez

This work was done as part of the research programme rCPS with project number 12697, financed by the Netherlands Organisation for Scientific Research (NWO).

Printed: ProefschriftMaken, The Netherlands.

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5325-9, NUR: 959.

Summary

Pipelining data-intensive sensing in control design: trading off control performance, robustness and processor usage

Data-intensive controllers use compute-intensive signal-processing algorithms to obtain sensing information. Signal-processing algorithms enable sensing features for which no simple sensor exists. Examples of such data-intensive controllers are found in Advanced Driver Assistance Systems (ADAS), Flexible Manufacturing Systems (FMS), or visual-servo control. In ADAS, image-processing algorithms are used to determine traffic road conditions; in FMS, image-processing algorithms allow to obtain (amongst others) the position of particular objects. Although data-intensive algorithms improve the sensing capabilities of the controller, they also introduce long time-varying sensing delay in the control loop. Therefore, the control design typically considers the worst-case of the sensing delay and the sampling period is selected to be longer than the worst-case delay. Long sensing delays produce long control sampling periods, which degrade the control performance. We define control performance in terms of time-domain metrics (e.g., settling time, overshoot) because it has an intuitive impact on system performance. For example, in FMSs long settling time results in a longer manufacturing time of each product, which decreases the overall system throughput. This thesis deals with the long sensing delay in data-intensive controllers, by exploring the design space of pipelined control systems. Pipelined control exploits the rise of multi-core architectures to implement parallel instances of the processing algorithm in a pipelined fashion. This produces extra sensing information that reduces the controller sampling period (while the sensing delay remains unaltered) potentially improving the control performance.

In pipelined control, each additional processing core potentially results in a better control performance, i.e., a shorter settling time. This opens up a trade-off between resource usage and control performance. The first contribution of this thesis is a strategy to analyse the resource-performance trade-off in pipelined control when constant (worst-case) delay is used. To do so, we propose a modelling and a control design technique. The modelling technique captures the dynamic model of the system to be controlled together with the number of parallel processing resources. The model can be used to design a well-known Linear Quadratic Regulator (LQR) controller. LQR has two tuning matrices that balance control effort and state deviation. However, choosing such matrices (i.e., LQR tuning) is difficult because the matrix values do not have an intuitive impact on time-domain controller performance metrics such as settling time. Additionally, in pipelined control each additional processing resource produces a larger pipelined model, which exponentially increases the number of parameters in the tuning matrices of an LQR, making a manual tuning impractical. The proposed control design technique uses Particle Swarm Optimization (PSO) to tune an LQR while maximizing the control performance (i.e., minimizing the settling time). Using these modelling and control design strategies, the trade-off between the number of resources and control performance can be analysed. This allows to choose a resource configuration that meaningfully improves the control performance.

In real-life systems, it is common to have continuous-time model uncertainties which might deteriorate the control performance. In the second contribution of this thesis, the trade-off analysis of the previous step is extended to include the robustness of the controller against model uncertainties. The technique first describes a strategy to discretize model uncertainties with one uncertain element in the input and state matrices. Next, the technique uses Lyapunov theory to quantify the controller robustness in terms of the maximum uncertainties that the previously designed pipelined control can tolerate before becoming unstable. The before mentioned trade-off is then extended with the quantified robustness. A resource configuration is chosen such that it meaningfully improves the quality of control while guaranteeing a minimum robustness against the model uncertainties.

The use of pipelined control requires constant availability of parallel processing resources to run the processing algorithm. Often, the processing resources are shared between the processing algorithm and other sporadic applications. The pipelined controller performance may then be further improved dynamically allocating the processing resources of the sporadic applications to the sensing algorithm. The third contribution of this thesis is a Reconfigurable Pipelined Controller (RPC). RPC design adjusts the hardware configuration (i.e., number of processing resources used for the sensing algorithm) depending on the number of sporadic applications and control performance requirements. The design of an RPC is divided into modelling and controller design. The dimension of a pipelined control model changes with the hardware configuration. RPC requires dynamic models with the same dimension for all hardware configurations under consideration. The proposed modelling strategy equalizes the size of the pipelined model for all hardware configurations used. The controller design requires one controller per hardware configuration. The controllers have to allow arbitrary switching between hardware configurations, so that whenever a new sporadic task needs to be executed the resources are freed. A controller with optimized performance is designed for the hardware configuration with the largest number of resources, because it produces the best performance among the hardware configurations. The remaining controllers are designed to provide stability guarantees when arbitrarily switching between hardware configurations. The resulting RPC performance is analysed with multiple run-time scenarios to provide design guidelines for the number of resources to be used.

Signal-processing algorithms typically are subject to variable delay due to scheduling policies, cache misses, or changes in data content. An application- and implementation-dependent histogram of delay can be used to identify the worst-case, the best-case, and one or more intermediate modes of execution time. Typically, the worst-case delay is unlikely to happen and larger than the delay modes. Modern processing resources are equipped with built-in timers that can be used to measure the actual execution-time on-line. The knowledge of the histogram of delay and the on-line measurability of the delay can be used to further improve the performance of a data-intensive controller. The fourth contribution of this thesis is an implementation-aware variable-delay pipelined control. We use the mentioned implementation-dependent histogram of delay to select sampling periods that potentially improve the control performance. The controller compensates for the variable delay using delayed sensed state vectors, employing on-line measurements of the delay and model-based predictions. Model-prediction accuracy highly depends on the model accuracy. Therefore, the controller performance is evaluated when prediction errors occur due to model uncertainty. A guideline is proposed to systematically select the control sampling period, number of pipes, and desired performance depending on the maximum expected model uncertainties.

In summary, this thesis proposes pipelined control to cope with the long sensing delay in data-intensive controllers. The contributions of this thesis explore the design space of pipelined control systems in terms of resource usage, control performance and robustness, taking into account delay variability. The techniques provide a designer the freedom to choose a control design strategy and a resource configuration suitable for the target application.

Contents

Summary **iii**

1 Introduction **1**
1.1 Embedded systems 2
1.2 Control systems 3
1.3 Embedded control systems 5
1.4 Data-Intensive Sensing Control (DISC) 7
1.5 State-of-the-art in DISC 8
1.6 Proposed design philosophy: pipelined-sensing control 9
1.7 Scientific challenges 10
1.7.1 Resource-aware modelling and control-design techniques 10
1.7.2 Robustness-aware design techniques 10
1.7.3 Variable processing resources 11
1.7.4 Variable sensing delay 11
1.8 Thesis objective and contributions 11
1.8.1 Resources vs QoC: modelling, QoC 12
1.8.2 Resources vs QoC trade-off under uncertainties 12
1.8.3 Efficient resource usage a via run-time reconfigurable controller 12
1.8.4 Implementation-aware variable-delay pipelined controller 13

2 Resources-QoC trade-off analysis **15**
2.1 Problem formulation 16
2.2 Motivational example: xCPS 17
2.3 Modelling pipelined controllers 19
2.3.1 Plant model 19
2.3.2 Sensing-to-actuating delay 20
2.3.3 Sampling period 21
2.3.4 Data-intensive sensor 21

2.3.5 Discretization of the pipelined model 22
2.3.5.1 Model discretization 22
2.3.5.2 Augmented non-delayed model 23
2.3.6 Alternative discretization of the pipelined model 24
2.3.6.1 Model discretization 24
2.3.6.2 Augmented non-delayed model 25
2.4 State-of-the-art LQR tuning 26
2.4.1 LQR controller 26
2.4.1.1 Control law 26
2.4.1.2 Design objectives 26
2.4.1.3 Feedback gain design 27
2.4.1.4 Feed-forward gain design 27
2.4.2 PSO concepts 29
2.4.3 Classical LQR tuning using PSO 30
2.4.3.1 Fitness metric 30
2.4.3.2 Algorithm flow 31
2.5 Proposed LQR tuning 33
2.5.1 Population definition 33
2.5.2 LQR tuning for a given number of pipes 34
2.6 LQR tuning: comparison 36
2.7 Trade-off analysis 37
2.7.1 Processing resources vs settling time 37
2.7.2 Settling time vs delay 38
2.7.3 Observations 38
2.8 Summary 41

3 Resources vs QoC trade-off analysis under model uncertainties 43
3.1 Problem formulation 44
3.2 Related work 45
3.3 Modelling and control design of a nominal system 46
3.3.1 Nominal model in pipelined control 47
3.3.2 Control design and trade-off analysis 48
3.4 Overview of the proposed design flow 49
3.5 Modelling pipelined systems with uncertainties 51
3.5.1 Continuous-time model with uncertainties 51
3.5.2 Discrete-time representation of continuous-time uncertainties . . 51
3.5.3 Discrete-time model with worst-case uncertainties 52
3.5.4 Augmented non-delayed model with uncertainties 53
3.5.5 Structure of the model uncertainties 53

3.5.6 Uncertainty bound . . . 54
3.5.7 Selection of D and E . . . 54
3.6 Model-uncertainty approximation . . . 54
3.7 Robustness of pipelined systems . . . 57
3.7.1 Preliminary lemmas . . . 58
3.7.2 Robustness analysis for performance-oriented controller . . . 58
3.7.3 Robustness-oriented pipelined controller design . . . 62
3.8 Robustness example of pipelined systems . . . 62
3.9 Feasibility study . . . 65
3.9.1 Overview . . . 65
3.9.2 Plant description and model . . . 66
3.9.3 HIL simulation . . . 68
3.9.3.1 Platform requirements . . . 68
3.9.3.2 CompSOC platform . . . 68
3.9.3.3 HIL set-up . . . 68
3.9.4 Application of the design flow . . . 69
3.9.5 HIL simulation results . . . 72
3.10 Summary . . . 74

4 Reconfigurable pipelined control 75
4.1 Preliminaries . . . 76
4.2 Problem definition . . . 76
4.3 Related work . . . 78
4.4 Modelling RPC . . . 79
4.4.1 Modelling the MC . . . 81
4.4.2 Modelling the RCs . . . 83
4.5 Control design in RPC . . . 85
4.5.1 Switching mechanism . . . 86
4.5.1.1 Rules and definitions for the switching mechanism . . . 87
4.5.1.2 Switching mechanism: from the MC to an RC . . . 89
4.5.1.3 Switching mechanism: from an RC to the MC . . . 92
4.5.2 Control design in the MC . . . 95
4.5.3 Control design in RC . . . 96
4.6 Summary of the design flow . . . 98
4.7 Case study: xCPS . . . 99
4.7.1 Assembly applications and resources . . . 99
4.7.2 Assembly process . . . 100
4.8 Simulation results . . . 101
4.8.1 IBC design . . . 101

4.8.1.1 Serial configuration SC 102
4.8.1.2 Pipelined configuration PC 102
4.8.1.3 Four-core RPC (RPC_4) 103
4.8.1.4 Five-core RPC (RPC_5) 103
4.8.2 Controller simulations 105
4.8.2.1 SC vs PC 105
4.8.2.2 PC vs four-core RPC 106
4.8.2.3 Four core RPC vs five core RPC 107
4.8.3 Discussion 108
4.9 Summary 108

5 Implementation-aware variable-delay pipelined control 109
5.1 Problem formulation 110
5.2 Related work 112
5.3 Variable-delay predictor controller 113
5.3.1 Run-time behaviour 114
5.3.2 Plant modelling 115
5.3.3 Controller block 116
5.3.4 Predictor block 116
5.3.5 Prediction length 118
5.3.6 Impact of prediction quality on QoC and control robustness . . . 118
5.3.7 QoC benchmarking 120
5.4 Variable-delay control design 120
5.4.1 Histogram characterization 120
5.4.2 Selection of actuation periods and resource configuration 121
5.4.3 Control design 122
5.4.4 QoC benchmarking to select the best controller 124
5.5 Case study: xCPS 125
5.5.1 Plant model 126
5.5.2 Image-processing algorithm 126
5.6 Simulation results 127
5.6.1 Histogram characterization 127
5.6.2 Selection of actuation periods and resource configuration 127
5.6.3 Control design 127
5.6.4 QoC benchmarking 128
5.7 Summary 131

6 Conclusions and future work 133
6.1 Conclusions 134

6.2 Future work 137

Bibliography **141**

List of Symbols **157**

Modelling and control design with pipelined sensing 157

PSO algorithm 158

Robustness Analysis 159

Reconfigurable pipelined control 160

Variable-Delay Control 161

List of Abbreviations **163**

Acknowledgments **165**

List of Publications **167**

Curriculum Vitae **169**

Chapter 1

Introduction

1.1 Embedded systems . . . 2
1.2 Control systems . . . 3
1.3 Embedded control systems . . . 5
1.4 Data-Intensive Sensing Control (DISC) . . . 7
1.5 State-of-the-art in DISC . . . 8
1.6 Proposed design philosophy: pipelined-sensing control . . . 9
1.7 Scientific challenges . . . 10
1.8 Thesis objective and contributions . . . 11

The introduction of technology in day-to-day life has allowed for continuous improvements in the quality of life of people by boosting their daily activities in terms of safety, efficiency, and comfort. Examples of these technologies range from home appliances such as Personal Computers (PCs) to more complex systems such as airplanes. Many modern technologies involve close interaction between physical parts (e.g., mechanical parts, environment) with cyber parts (e.g., embedded systems, software and communication). Such technologies are commonly referred to as Cyber-Physical Systems (CPSs). A CPS can be a home appliance such as washing machines as well as industrial systems such as flexible manufacturing systems, automotive systems, aerospace systems, robotic systems, and many others [103, 72]. An example of a CPS is shown in Fig. 1.1. Designing a CPS is a complicated task since it requires to consider a wide range of system specifications such as performance, reliability, safety, flexibility, and cost-effectiveness. For example, in Flexible Manufacturing Systems the perfor-

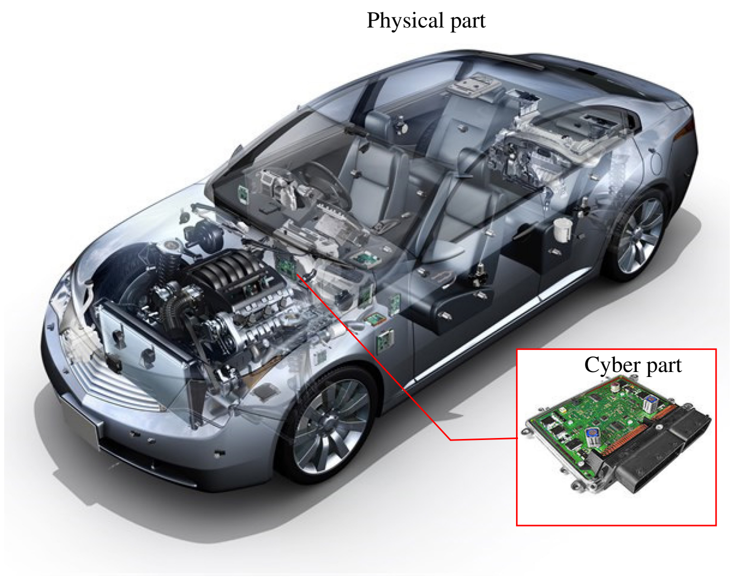


Figure 1.1: Example of a CPS. A car has one or more embedded systems (cyber part) that interact with the car components (physical part) such as motor, transmission, or navigation. Source of figure [7].

mance can be specified as output products per time unit, the reliability as number of failures per year, and the cost-effectiveness as the ratio between system productivity and system cost. To design such CPSs, multiple engineering disciplines are involved because of the tight interaction between physical and cyber aspects. Some key disciplines include electronics, mechanics, embedded systems, control systems, communications and networking. Designing a CPS requires co-design and integration strategies, that take into account the interplay across multiple disciplines to satisfy the specifications of the CPS [79, 108]. In this thesis, we focus on the challenges arising from the co-design of control systems and embedded systems in CPSs.

1.1 Embedded systems

An embedded system is a computational system that regulates the behaviour of a larger system such as a CPS [57]. Similar to a PC, an embedded system is composed of one (or more) processors(s), a set of memories (e.g., cache, RAM, ROM), and a set of peripherals (e.g., serial and parallel data interfaces, digital or analogue inputs and outputs). One of the differences between an embedded system and a PC is that the former is (commonly) used within a larger system (e.g., a CPS), while the latter is used by an end user.

The functionality of an embedded system is defined by embedded software. Such embedded software is designed to meet the CPS specifications such as performance, reliability, and cost-effectiveness. Therefore, the design of this software must guarantee the desired behaviour of the CPS, as well as timing requirements which are usually specified in terms of deadlines, throughput, jitter, and periodicity. These timing requirements are either hard or soft. Hard timing requirements are assigned to applications whose satisfaction is critical for the CPS specifications (e.g., performance, reliability, safety), while soft timing requirements are assigned to non-critical tasks (e.g., user interfaces) [114, 59, 146]. The goal of an embedded-system engineer is to ensure that such requirements are satisfied within the capabilities of the platform: processing power, memory capacity, and so on. In this thesis, we deal with a type of applications with (typically) hard timing requirements called control systems, which are explained in the next section.

The capabilities of an embedded system have been growing according to Moore's Law prediction [97]. Moore's Law predicted that the number of transistors in integrated circuits will double every two years. For decades, this enabled substantial performance improvement in embedded systems with a single processor unit by including faster and denser components, such as memories and processors. Nowadays, the performance improvement of an embedded system with a single processor is limited due to the excessive heat dissipation and power consumption of the processor [43, 49]. The current trend in the embedded-systems domain is to provide parallel processing (i.e., multiprocessing) units to further enhance the processing capabilities of an embedded system (see for example Fig. 1.2). Parallel processing architectures are easier to cool down because the processors are simpler and use fewer transistors, which results in lower heat dissipation and power consumption [43, 16, 49]. Examples of embedded systems with parallel processing capabilities are multi-core processors, FGPAs, and GPUs.

In the scope of this thesis, we consider control systems with hard timing requirements implemented in embedded systems with multiprocessing capabilities.

1.2 Control systems

A control system is an interconnection of components (e.g., amplifiers, capacitors, resistors, embedded systems) that drives a physical process (i.e., a plant) towards a de-

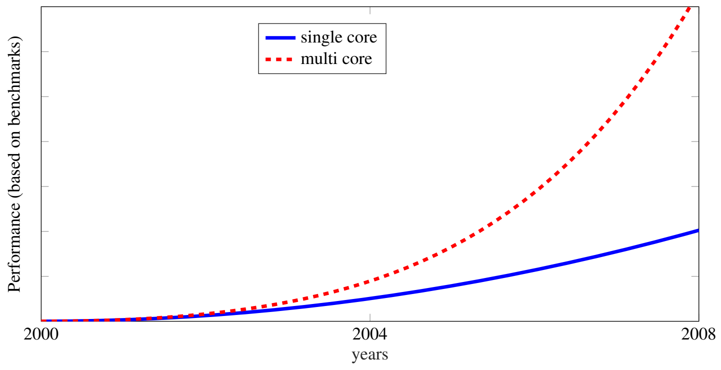


Figure 1.2: Performance comparison of embedded-system architectures. Figure adapted from [43].

sired reference [33, Chapter 1]. A schematic of a control system is shown in Fig. 1.3. The control system is connected to a sensor and an actuator. The sensor converts the state vector (e.g., the plant outputs) into electrical signals; the actuator influences the state vector based on the controller input. A control-system engineer deals with the design of controllers (i.e., control laws) that satisfy a performance metric referred to as Quality of Control (QoC). Examples of performance metrics are time-domain metrics such as settling time, tracking velocity or overshoot (e.g., [75, 76, 1]), Quadratic Costs (e.g., [35, 9, 126]), the closed-loop pole locations [127], or robustness guarantees (e.g., [40, 56]). Time-domain metrics are preferred in real-time applications since they provide intuitive information of how QoC affects CPS-level design specifications such as performance and reliability. For example, a common QoC metric is settling time, which corresponds to the time that the controller takes to drive the plant output from an initial state to a desired reference. Fig. 1.4 compares the settling time of two different plants. Note that the plant with the shorter settling time arrives to the reference first. A better QoC (i.e., a shorter settling time) can increase the performance of a CPS since it implies that the plant is driven faster to the reference, which can be translated into a higher system productivity. In this thesis, we consider settling time as a performance metric for computing QoC.

A control system can be implemented in the analogue domain using operational amplifiers, capacitors, resistors and inductors, or in the embedded domain using an embedded system. embedded-systems implementations are nowadays preferred because they provide advantages such as ease of implementation and upgrading complex

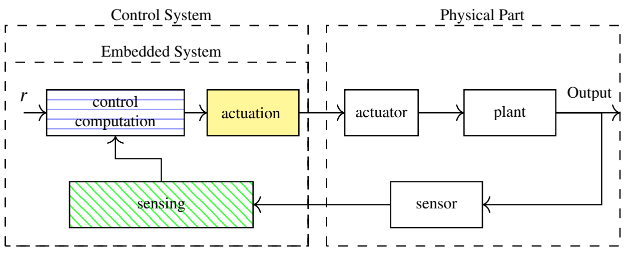


Figure 1.3: Example of (embedded) control system. $r \in \mathbb{R}$ is the controller reference. The actuation, control computation, and sensing blocks represent the tasks execution in the embedded system.

calculations, capability of storing data, and precision that is not affected by the drift of components over time [104, Chapter 1.2]. A control system implemented on an embedded system is referred to as embedded control system, which is the scope of this thesis.

1.3 Embedded control systems

An embedded control system executes three main tasks: sensing, control computation, and actuation. Fig. 1.3 shows a schematic of an embedded control system. The sensing task translates the sensor measurements into data that the controller can interpret. For example the pulses of a quadrature encoder are translated into a shaft position by the sensing task. The control computation task computes a controller (i.e., a control law) that drives the plant towards the reference. The actuation task translates the computed control law into a physically realizable signal for the actuator. For example in a motor control system, the controller input is translated into a PWM duty cycle by the actuation task. The implementation of these tasks in an embedded system must guarantee two timing properties: the sampling intervals and the sensing-to-actuating delay. The sampling intervals represent the time elapsed between two consecutive sampling actions. The sampling intervals are (commonly) assumed to be equal resulting in periodic sampling; therefore it is referred to as the sampling period. The sensing-to-actuating delay represents the time elapsed between the start of the sensing task and the end of the corresponding actuation task. The sensing-to-actuating delay is commonly assumed negligible compared to the sampling period to simplify the control design. An example of these timings is shown in Fig. 1.5. Note that the sampling period is much larger than the sensing-to-actuating delay.

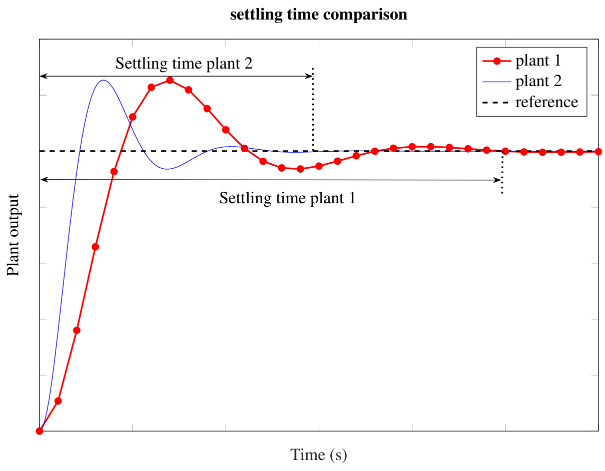


Figure 1.4: Settling time example. The settling time is the time that a plant takes to reach a band within 2% of the reference. Plant 2 has a settling time shorter than Plant 1 since it reaches the 2% band faster.

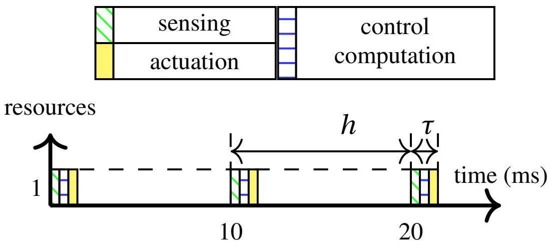


Figure 1.5: Ideal implementation of an embedded control system. With $h \in \mathbb{R}^+$ the sampling period and $\tau \in \mathbb{R}^+$ the sensing-to-actuating delay. Notice that $h >> \tau$.

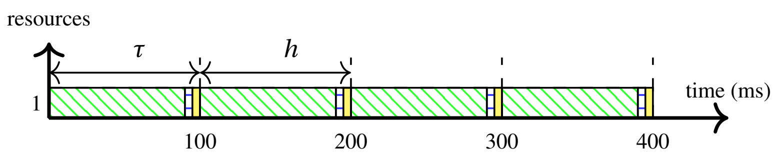
The design of an embedded control system typically employs the principle of separation of concerns [99]. This principle states that the control design can be done independently of the platform where the controller is going to be implemented, e.g., the sensing-to-actuating delay is negligible compared to the sampling period (see Fig. 1.5); the embedded engineer has to guarantee that the timing requirements of the controller are satisfied in the embedded platform [99]. To do so, an option is to use embedded platforms with sufficiently fast processing capabilities, i.e., over-dimensioned computational resources. However, over-dimensioning might unnecessarily increase the CPS cost, which negatively affects the cost-effectiveness of the system. To improve the resource usage, an alternative approach is to co-design the controller taking into account the resulting timing properties on the embedded system, e.g., sampling period, delays, jitter [122, 123]. In such a co-design strategy, the sampling period of the controller must be chosen equal to or larger than the resulting sensing-to-actuating delay. This gives enough time for the embedded system to compute the controller tasks before a new sensing task begins (see Fig. 1.5). In this thesis, we consider co-design strategies for embedded control systems.

In some control applications, the sensing task of the controller requires the computation of complex algorithms, which enlarges the sensing-to-actuating delay. When the sensing-to-actuating delay is long, the sampling period of the controller is inevitably enlarged, which is a known cause of QoC deterioration [110]. For example, Fig. 1.6a shows an embedded control system where the sensing task takes considerably longer time to execute than the control computation and actuation tasks, which imposes a long sampling period. This thesis considers embedded control systems with a long sensing delay and consequently, a long sampling period. Such controllers are referred to as Data-Intensive Sensing Control (DISC).

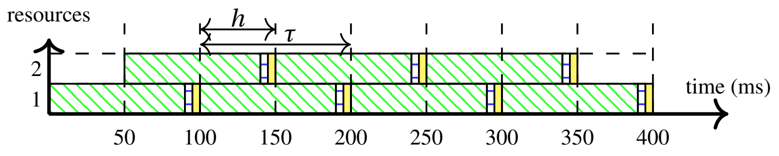
1.4 Data-Intensive Sensing Control (DISC)

DISC is a type of controller where the sensing task runs signal-processing algorithms. The main advantage of using signal-processing algorithms is the capability of generating sensing information that is not easily acquired by regular sensors. A common example of DISC is Image-Based Control (IBC), which is found in Advanced Driver Assistance Systems (ADAS) [67, 68] and visual servo control [22]. For example, in ADAS the image-processing algorithm allows to determine traffic signals or road conditions, while in visual servo control it allows to acquire the position of particular objects.

The use of signal-processing algorithms as a sensing strategy comes at a cost of additional processing resources and a longer sensing delay (caused by the processing



(a) Serial DISC implementation. $h = \tau$.



(b) Pipelined DISC implementation with two sensing cores. $h = \frac{\tau}{2}$.

Figure 1.6: Example implementations of DISC. A shorter sampling period h is achieved by pipelined control. The colours follow those of Fig. 1.5.

algorithm) in the control loop. The additional processing resources of a DISC can have a negative impact on the cost-effectiveness design specification of the CPS. The long sensing delay enlarges the controller sampling period, which causes control problems such as QoC degradation, robustness problems, and even control instability [110]. These problems can negatively influence the CPS specifications such as performance, reliability, flexibility and safety. For example, in visual servo control a low QoC (i.e., long settling time) slows down the movement of a robotic actuator, which can negatively affect the system productivity (i.e., performance); likewise, an unstable controller in visual servo control forces the robotic actuator to stop, which affects its reliability. This thesis addresses the design of DISC to cope with the aforementioned challenges.

1.5 State-of-the-art in DISC

The problem of a long sensing delay has been investigated in the literature from both the embedded-systems and the control-systems viewpoints. Examples of control system focused approaches are found in [37, 66, 69, 133, 132]. They use estimation techniques based on Kalman filters to interpolate the state vector between sensing instances [37, 66] or multi-rate strategies that allow an actuation period shorter than the sensing delay [69, 133]. In the embedded-systems domain, the approaches focus on reducing the image-processing delay by creating faster parallel implementations of the algorithms in specialized embedded platforms such as GPUs [5, 124] or FPGAs [71, 142]. However, these approaches have some limitations. For example, the control-systems strategies rely on the system model to estimate sensing information. Such estimations are vulnerable to modelling errors, they are sensitive to unmodelled disturbances, and their prediction errors increase with longer delays. The embedded-systems approaches rely on parallelization. Parallelization is not applicable to all real-life algorithms. Additionally, it may be time consuming to model the algorithm to identify the scope of parallelism, while the resulting delay might not be shorter than the desired sampling period of the plant.

An alternative to cope with long sensing delays is to combine knowledge from both control-systems and embedded-systems domains to create a co-design strategy known as pipelined-sensing control. This thesis focuses on the design of such pipelined controllers.

1.6 Proposed design philosophy: pipelined-sensing control

Fig. 1.6 compares a classical (i.e., serial) implementation with a pipelined implementation with a 2-core resource configuration (i.e., with two sensing cores). The idea is that the additional processing resource increases the update frequency of the sensing information while the sensing delay remains the same. These extra updates of the sensing information can be used by the controller to improve QoC. A pipelined-sensing controller requires an embedded system with sufficient parallel processing resources to compute the sensing algorithm in a pipelined fashion instead of a serial fashion. It also requires that subsequent samples can be independently processed by the sensing algorithm.

Pipelined-sensing control was introduced by Krautgartner and Vincze in [75] and extended in [76, 23, 144]. [75, 76] compared a serial and a pipelined sensing in visual servo-positioning systems using PID controllers. The pipelined sensing is shown to outperform the serial sensing when comparing steady-state error in a tracking application. [23] compared a PID control, a P controller with a Kalman filter in the sensing, and a Generalized Predictive Controller (GPC) in a vision-based control application. The result shows that GPC achieves the best performance in terms of real-time performance metrics (e.g., Integral of Absolute Error). [144] studied pipelined controller in the context of networked control systems. The authors tested PID controllers showing that pipelined control outperforms serial control when comparing settling time, overshoot, and phase margin. Pipelined control has not been studied in the scope of modern embedded systems with multiprocessing capabilities. Control-design strategies are still to be adapted to pipelined systems such that the resource usage is taken into consideration while QoC improvement and control robustness are achieved. Such requirements can have an impact on system-level requirements of CPS such as performance, reliability, and cost-effectiveness.

1.7 Scientific challenges

In the design of pipelined-sensing controllers with modern multiprocessing embedded systems, we have identified the following challenges.

1.7.1 Resource-aware modelling and control-design techniques

In pipelined control, the use of additional sensing resources results in a better QoC leading to a resource vs QoC trade-off. Analysing such a trade-off allows to minimize the used resources while still improving the QoC. Such an analysis can further be used to improve the cost-effectiveness of a CPS, as well as its performance. To analyse such a trade-off, a modelling and a control-design technique are required. The modelling technique has to capture the interplay between controlled plant dynamics and processing resources. The control-design technique needs resource-awareness to provide controllers with improved QoC in terms of time-domain metrics (e.g., settling time).

1.7.2 Robustness-aware design techniques

In modern CPSs, it is common to have continuous-time models with a known margin of inaccuracy i.e., the model includes model uncertainties. These uncertainties of the model cause QoC deterioration, which in turn might affect the resources QoC trade-off of pipelined controllers, the system performance, the system cost-effectiveness, and the system reliability. To analyse the impact of such uncertainties in pipelined systems, a modelling and a robustness-analysis technique are required. The modelling technique must capture the interplay between system dynamics, the processing resources, and the uncertainties. The robustness analysis technique must quantify the controller robustness against the aforementioned model uncertainties.

1.7.3 Variable processing resources

In CPSs, the processing resources are often shared between the DISC and other applications. Modern embedded platforms are commonly equipped with operating systems that dynamically allocate processing resources to different tasks. Dynamically allocating (i.e., reconfiguring) the processing resources to the pipelined controller can be used to improve QoC (in terms of time-domain metrics), CPS performance, and the CPS cost-effectiveness. Such reconfigurability requires a dedicated modelling technique and a control-design technique. The modelling technique needs to capture the interplay between variable processing resources and controlled plant dynamics. The control-design technique must cope with the variability in the model, must improve the QoC, and must provide stability for all the resource configurations.

1.7.4 Variable sensing delay

The implementation of a signal-processing algorithm in modern embedded systems typically results in a variable delay, due to platform-related variation (e.g., cache misses) or algorithm-related variation (e.g., number of regions of interest). Such a variable delay has a worst-case delay which is significantly longer than the more common delays. State-of-the-art design techniques assume a constant (i.e., worst-case) sensing delay in the controller design. Considering the most common delays in the control design (instead of the worst-case delay) can be used to further improve the QoC, the performance, and the cost-effectiveness. To do so, a variable-delay control-design strategy for pipelined systems is required. Typical design techniques for variable-delay control provide robustness guarantees against the variable-delay while the QoC is not measured in terms of time-domain metrics. Therefore, a design technique for pipelined systems must not only cope with the variability of the sensing delay, but also provide QoC improvement (in terms of time-domain metrics) compared to a worst-case design.

1.8 Thesis objective and contributions

The main objective of this thesis is to provide design techniques for pipelined-sensing controllers using modern multiprocessing embedded systems. The design techniques must improve on QoC, guarantee control robustness, and take into account resource usage. A CPS designer can use these techniques to guarantee system-level specifications such as performance, cost-effectiveness, and reliability. The contributions of this thesis are subdivided as follows.

1.8.1 Resources vs QoC: modelling, QoC optimization and trade-off analysis (Chapter 2)

The first contribution of this thesis is aligned with Challenge 1.7.1. We propose a modelling and QoC optimization technique that allows to analyse the trade-off between processing resources and QoC in pipelined-sensing control. For a given resource configuration, a modelling technique is adapted from the literature such that the interplay between processing resources and plant dynamics is captured. A control-design technique based on Particle Swarm Optimization (PSO) is also proposed. The technique finds the tuning parameters of the well-known Linear Quadratic Regulator (LQR) which produces a controller with minimum settling time. The modelling and control-design techniques are applied to all available resource configurations to analyse the resources-QoC trade-off. Guidelines are then provided to choose a resource configuration that meaningfully improves the QoC.

This chapter is an extended version of [91].

1.8.2 Resources vs QoC trade-off under model uncertainties (Chapter 3)

The second contribution of this thesis is aligned with Challenge 1.7.2. We extend the trade-off analysis between processing resources and QoC to include robustness under model uncertainties. Our contribution introduces a method to benchmark discrete-time model uncertainties based on continuous-time uncertainties with one uncertain element in the state and input matrices. The discrete-time uncertainties are used to quantify the robustness of a designed controller, or to design a controller that enhances robustness. The trade-off analysis mentioned above is then extended including a desired robustness for the controller. A resource configuration that meaningfully improves the QoC while guaranteeing a robustness constraint can then be selected.

This chapter is based on publication [92].

1.8.3 Efficient resource usage via run-time reconfigurable controller (Chapter 4)

The third contribution of this thesis is aligned with Challenge 1.7.3. We propose a Reconfigurable Pipelined Controller (RPC) that allows a run-time allocation of processing resources based on the CPS needs. The design of an RPC is divided into a modelling and a control-design technique. The modelling technique is adapted from the literature to capture the interplay between variable processing resources and plant dynamics. The control technique designs a set of controllers for each available resource configuration, such that QoC and stability are guaranteed. This control-design strategy allows the RPC to arbitrarily switch the processing resources at run-time to allocate processing resources to other tasks running in the system. The CPS can then more efficiently allocate extra processing resources to either the RPC or to other tasks depending on the system needs.

This chapter is based on publication [90].

1.8.4 Implementation-aware variable-delay pipelined controller (Chapter 5)

The fourth contribution of this thesis is aligned with Challenge 1.7.4. We propose an implementation-aware variable-delay pipelined controller. The controller design uses the delay modes to select sampling periods that potentially improve the QoC compared to a worst-case (fixed delay) design. The controller compensates for the variable delay using on-line measurements of the sensing delay, the delayed pipelined measurements, and model-based predictions. Such model-based predictions are highly sensitive to model mismatches. Therefore the QoC is benchmarked using the maximum expected level of model uncertainties and compared with a QoC produced by a worst-case design. Guidelines are then provided to select a sampling period that improves the QoC given the uncertainty levels.

This chapter is based on publication [88].

Combining the contributions of this thesis allows to design pipelined-sensing controllers that not only improve the QoC but also consider the resource usage and control robustness. These contributions can be further used to guarantee design specifications in CPS such as performance, reliability, and cost-effectiveness.

Chapter 2

Resources-QoC trade-off analysis

2.1 Problem formulation 16
2.2 Motivational example: xCPS 17
2.3 Modelling pipelined controllers 19
2.4 State-of-the-art LQR tuning 26
2.5 Proposed LQR tuning 33
2.6 LQR tuning: comparison 36
2.7 Trade-off analysis 37
2.8 Summary 41

To cope with a long sensing delay in a Data-Intensive Sensing Control (DISC), we use pipelined-sensing control. Pipelined-sensing control uses extra processing resources to compute the sensing algorithm in a pipelined fashion to reduce the sampling period. A shorter sampling period can potentially lead to a better control performance. Therefore, there is a trade-off between processing resources and Quality of Control (QoC) in pipelined control. In this chapter, we present a method to analyse this trade-off between processing resources and QoC in pipelined-sensing control, which shows the impact of processing resources in control performance. To do so, we model the plant taking into account a resource configuration (i.e., the number of processing resources) to be used. For such a model and resource configuration, we design a controller with optimized QoC using well-known control-design techniques. The modelling and control design is repeated across multiple resource configurations to explore the before mentioned trade-off space. We provide design guidelines based on our observations.

This chapter is an extended version of the work published in SAC 2017 [91].

2.1 Problem formulation

To analyse the resources-performance trade-off a in pipelined-sensing control, a modelling technique and a control-design technique are required for each resource configuration to be considered. The modelling technique is required to integrate the interplay between the plant dynamics and the processing resources, while the control-design technique is required to find a controller that maximizes control performance (i.e., the QoC) for a given resource configuration. We propose both modelling and control-design techniques. Our modelling technique is an adaptation of state-of-the-art models such as [104, Chapter 2]. Our control-design technique finds a controller that maximizes QoC. The settling time is selected as a performance metric, i.e., $QoC = S_t^{-1}$ with S_t the settling time, as explained in Section 1.2. A shorter settling time implies a better QoC. As a controller, we use the well-known Linear Quadratic Regulator (LQR) [80]. Although many other control-design techniques might be applicable here, this technique suffices to show the impact of processing resources on control performance. An LQR minimizes a quadratic cost derived from the state vector and the controller input. The state deviation and the control input are weighted using the positive semi-definite matrix Q and positive-definite matrix R, respectively. Choosing such matrices is referred to as LQR tuning. The resulting controller balances the control input and the state deviation, which does not necessarily imply a shorter settling time. We present a strategy to tune an LQR such that the settling time is minimized in a pipelined-sensing control. Next, we use the tuned controller to analyse the aforementioned trade-off.

Most of the strategies for LQR tuning focus on finding the diagonal elements of Q and R, because a diagonal matrix remains positive (semi) definite if its elements are greater than or equal to zero. Examples of such methods include trial and error [106], Bryson [20], and evolutionary algorithms [35, 55, 100, 101, 127]. Bryson proposed a relationship between the maximum desired value of the states, the controller input, and the diagonal elements of the Q and R matrices. The resulting matrices are manually tuned till the desired QoC is met. Among the evolutionary algorithms, Particle Swarm Optimization (PSO) has shown superior performance capabilities over genetic algorithms [55, 100]. PSO was used to tune the diagonal elements of Q and R using the classical PSO approach in [35, 127] and using modified PSO approaches in [55, 101]. In [52], PSO was used to tune not only the diagonal but also the off-diagonal values of Q and R. However, the design procedure does not allow to explore the design space, while respecting the positive (semi) definiteness of the tuning matrices. Exploring all the elements of Q and R, while respecting positive (semi) definiteness remains an open challenge. Our method to analyse the trade-off shows that tuning all the elements of Q and R using PSO potentially improves the settling time compared to tuning only the diagonal using the classical PSO approach. Note that the modified PSO can also be used to compare with our method. However, classical PSO is better benchmarked in the literature and hence, is used in this work.

Our contributions in this chapter are three fold: (i) we adapt the modelling technique of [104, Chapter 2] to capture the interplay between processing resources and the dynamic system in pipelined systems. (ii) We propose a systematic PSO-based LQR tuning method for tuning all elements of Q and R matrices respecting the positive (semi) definiteness of Q and R. The proposed LQR tuning shows improved QoC over the state-of-the-art tuning methods that consider the diagonal of Q and R. (iii) Using the LQR tuning, we are able to optimize the QoC for a given processing configuration. We characterize the relation and trade-off between processing delay, the number of sensing pipes and the QoC.

2.2 Motivational example: xCPS

The eXplore Cyber-Physical Systems (xCPS) platform is an industrial assembly-line simulator shown in Fig. 2.1, which is used for teaching and research purposes [3, 2]. The machine assembles or disassembles circular complementary pieces (i.e., assembly pieces) that come in two shapes: lower and upper parts.

We evaluate our approach through simulations considering characteristics and requirements of xCPS. In particular, the assembly section of xCPS is considered in our simulations. In such an assembly section, conveyor belts move the assembly pieces through the machine actuators: a turner, multiple stoppers, and a Pick-and-Place unit, among others. Regulating the speed and positions of the assembly blocks is crucial for guaranteeing the correctness of the assembly process. To do so, an image-processing algorithm based on the Hough transform for circles is used to measure the speed and position of the assembly blocks [145, 31, 6]. A DISC uses the output of this algorithm to regulate the assembly process. Note that the QoC of the DISC directly affects the machine performance: a shorter settling time results in assembly blocks travelling faster which potentially increases the machine throughput. Improving the QoC is therefore meaningful for xCPS.

Example 2.1. ***Motivational example:*** *To illustrate the benefits of pipelined control, consider that in xCPS the image-processing algorithm introduces a sensing-to-actuating delay of* $\tau = 84$ ms*, the conveyor belt motor speed has an open-loop settling*

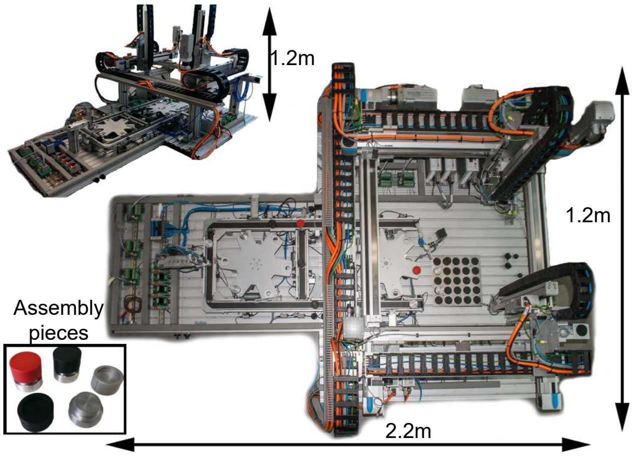
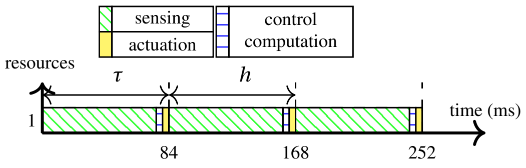
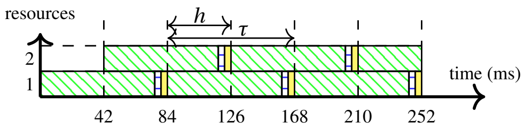


Figure 2.1: xCPS platform [3].



(a) Serial DISC implementation.



(b) Pipelined DISC implementation with two sensing cores.

Figure 2.2: Example implementations of the DISC in Example 2.1.

time of 2.80 *s (i.e., the motor speed stabilizes* 2.80 *s after applying a change in the input voltage), and the two resource configurations of Fig. 2.2 are used. Fig. 2.3 shows an example of the QoC achieved by Image-Based Controls (IBCs) with both resource configurations. The pipelined controller shows a settling time significantly shorter than the serial implementation. Notice that the plant response remains at zero till the* *elapsed time is longer than τ. This is because the image-processing algorithm has yet not delivered the first sensing information; therefore no sensing information is available to the controller.*

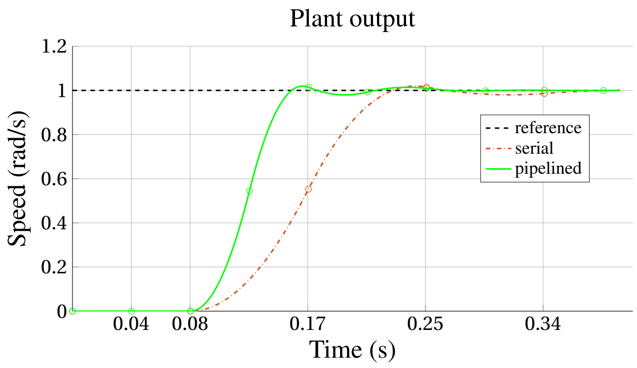


Figure 2.3: Example of controller response of Example 2.1 with and without pipelining. The round markers denote sampling instants.

2.3 Modelling pipelined controllers

As described in Section 1.6, a pipelined controller uses additional processing resources to increase the sensing information available to the controller. In the next subsections, we model such a system.

2.3.1 Plant model

We consider the following single-input single-output linear time-invariant systems of the form:

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t), \end{aligned} \tag{2.1}$$

where $A_c \in \mathbb{R}^{n\times n}$ the state, $B_c \in \mathbb{R}^{n\times 1}$ the input, and $C_c \in \mathbb{R}^{1\times n}$ are the output matrices. Further, $x(t) \in \mathbb{R}^{n\times 1}$ is the state vector, $u(t) \in \mathbb{R}$ is the input, and $y(t) \in \mathbb{R}$ is the output at time $t \in \mathbb{R}^{\geq}$. The plant has $n \in \mathbb{Z}^{+}$ states, 1 input, and 1 output. In pipelined control, there is a sensing-to-actuating delay introduced by the latency of the data-intensive algorithm. Therefore, the control input depends on old state-vector information, i.e.,

$$u(t) = f(x(t-\tau))$$

where $\tau \in \mathbb{R}^+$ is the sensing-to-actuating delay.

2.3.2 Sensing-to-actuating delay

Sensing, control computation and actuation operations are executed on an embedded platform. They take a finite time to execute, which may vary over different executions. We assume the operations run on a state-of-the-art predictable multi-core platform (such as [48]) where worst-case execution time is measurable and predictable. These tasks have worst-case execution time $\tau_s \in \mathbb{R}^+$, $\tau_c \in \mathbb{R}^+$ and $\tau_s \in \mathbb{R}^+$ for sensing, control computation and actuation operations, respectively. Since the sensing operation involves computationally intensive image processing, in DISC, τ_c and τ_a are short compared to τ_s, i.e.:

$$\tau_s \gg \tau_c + \tau_a.$$

This property holds for many real-time systems, where a signal-processing algorithm is used to acquire sensing data. The sensing-to-actuating delay τ is given by,

$$\tau = \tau_s + \tau_c + \tau_a. \tag{2.2}$$

Note that a long τ_s increases τ. A large τ means that pipelining might be beneficial. Sensing, control computation, and actuation operations are activated in a time-triggered fashion. The start times of the k^{th} sensing operation $t_k^s \in \mathbb{R}^{\geq}$ are defined by

$$t_k^s = kh,$$

with $h \in \mathbb{R}^+$ the sampling period measured between the start of two consecutive sensing operations, i.e., $h = t_{k+1}^s - t_k^s$, and $k \in \mathbb{Z}^{\geq}$ the discrete-time index. The start time of the k^{th} control computation $t_k^c \in \mathbb{R}^+$ and the k^{th} actuation operation $t_k^a \in \mathbb{R}^+$ are given by,

$$\begin{aligned} t_k^c &= t_k^s + \tau_s \\ t_k^a &= t_k^c + \tau_c. \end{aligned}$$

The corresponding actuation operations are therefore completed at $t = t_k^a + \tau_a$. Since τ_s,τ_c, and τ_a are assumed constant, the sensing-to-actuating delay τ is also constant with the above time-triggered activation policy.

2.3.3 Sampling period

The sampling period is the time elapsed between the start of two consecutive sensing operations. In an ideal case, τ is negligible compared to the plant dynamics. In this case, the sampling period can be chosen, for example, using the following rule of thumb [104, Chapter 2.9]:

$$h_{Rt} \approx \frac{R_t}{10}, \tag{2.3}$$

with $h_{Rt} \in \mathbb{R}^+$ is sampling period based on the rise time and $R_t \in \mathbb{R}^+$ the plant open-loop rise time. However in DISC, the sensing task introduces significant sensing-to-actuating delay τ. Therefore, the sampling period is chosen according to the execution times of the platform operations. Using the serial implementation of Fig. 2.2a, the sampling period is defined as:

$$h_s = \tau, \tag{2.4}$$

where $h_s \in \mathbb{R}^+$ is the serial sampling period in a DISC. Therefore, the sensing-to-actuating delay may force a sampling period which is larger than the ideal one computed based on the plant dynamics, i.e., $h_s > h_{Rt}$. This potentially limits the controller performance.

Pipelining the sensing algorithm is a solution for this performance limitation. In pipelined-sensing control, the sampling period is defined not only by the execution times of operations, but also by the number of used processing resources:

$$h = \frac{\tau}{\gamma}, \tag{2.5}$$

where $\gamma \in \mathbb{Z}^+$ is the number of processing resources. Note that increasing γ produces a smaller sampling period, which potentially improves the QoC. The duration τ is divided into γ samples of length h. Given that the sensing operation dominates the sensing-to-actuating delay, the corresponding control and actuation operations are typically performed in the last interval of length h. These timings are illustrated in Fig. 2.2b. Recall from Section 1.6 that we assume that subsequent samples can be processed independently.

2.3.4 Data-intensive sensor

The data-intensive sensor is the device that acquires the data for the sensing operation. In this work, we consider a camera as a data-intensive sensor. For a controller, it is important that a sensing operation is started periodically, according to the sampling period h. Often, cameras are triggered at a fixed frame rate. We consider a time-triggered camera [64] which allows for acquisition of frames in a time-triggered fashion. For the controllers designed in this chapter (and Chapters 3 and 5), it suffices that the data-intensive sensor acquires frames periodically with a period given by:

$$h_{ac} = h, \tag{2.6}$$

where $h_{ac} \in \mathbb{R}^{+}$ is the period of data acquisition.

2.3.5 Discretization of the pipelined model

2.3.5.1 Model discretization

The discrete-time equivalent of Eq. 2.1 has the following form:

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_{k-\gamma} \\ y_k &= C_c x_k, \end{aligned} \tag{2.7}$$

where $A_d \in \mathbb{R}^{n\times n}$ and $B_d \in \mathbb{R}^{n\times 1}$ are the discrete-time state and input matrices respectively, defined as [8]:

$$\begin{aligned} A_d &= e^{A_c h} \\ B_d &= \int_0^h e^{A_c s} B_c ds. \end{aligned} \tag{2.8}$$

$x_k \in \mathbb{R}^{n\times 1}$ and $y_k \in \mathbb{R}$ are the discrete-time state and output vectors, respectively, and $x_k := x(kh)$ and $y_k := y(kh)$ with $k \in \mathbb{Z}^{\geq}$. $u_{k-\gamma} \in \mathbb{R}$ is the discrete-time control input which is designed in Section 2.4.1. The control input is implemented using a Zero-Order Hold (ZOH) to keep the actuation signal constant between consecutive sampling instants. This results in $u(t) := u_{k-\gamma}$ for all $t \in [kh, (k+1)h)$. The continuous-time control input is therefore piecewise constant in this interval. Fig. 2.4 illustrates the relationship between different timing components. Note that due to the sensing-to-actuating delay, the first available sensing information of the model of Eq. 2.7 appears at time $t = \tau_s$ (assuming that the first sensing operation is triggered at $t = 0$). Therefore, the control inputs in the time interval $[0, \tau)$ have no sensing information. This is addressed in the control input design presented in Section 2.4.1.

In the model of Eq. 2.7, $u_{k-\gamma}$ captures the sensing-to-actuating delay assuming that the control input is delayed $\tau = \gamma h$ time units. Capturing the sensing-to-actuating delay can also be done by delaying the observation of the state vector, or in other words, by only using information that is $\tau = \gamma h$ time units old. This results in a model closer to the physical behaviour of the system, as is shown in Section 2.3.6. However, the model of Eq. 2.7 reduces the sizes of the model matrices compared to the alternative model of Section 2.3.6, which is why it is the preferred approach in this thesis.

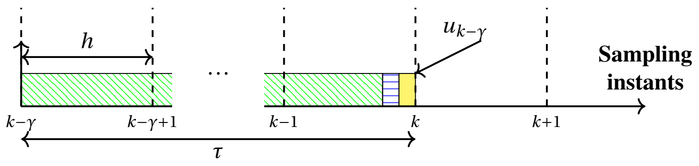


Figure 2.4: Relationship between the delay τ and the sampling period h. The colour notations are the same as in Fig. 2.2b.

2.3.5.2 Augmented non-delayed model

The model of Eq. 2.7 uses a delayed control input $u_{k-\gamma}$. We transform this model into a standard non-delayed canonical form, which can be used to apply a wide range of analysis and design methods. To do so, the following state-vector augmentation is proposed:

$$z_k = \begin{bmatrix} x_k^T & u_{k-\gamma} & u_{k-\gamma+1} & \cdots & u_{k-2} & u_{k-1} \end{bmatrix}^T, \tag{2.9}$$

which results in the following discrete-time model of pipelined systems:

$$\begin{aligned} z_{k+1} &= \Phi_d z_k + \Gamma_d u_k \\ y_k &= C_d z_k, \end{aligned} \tag{2.10}$$

with $z_k \in \mathbb{R}^{(n+\gamma)\times 1}$ the discrete-time augmented state vector, $u_k \in \mathbb{R}$ the discrete-time input, $y_k \in \mathbb{R}$ the discrete-time output with $y_k = y(kh)$ and $k \in \mathbb{Z}^{\geq}$, $\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ and $\Gamma_d \in \mathbb{R}^{(n+\gamma)\times 1}$ the augmented discrete-time state and input matrices, respectively, and $C_d \in \mathbb{R}^{1\times(n+\gamma)}$ the discrete-time augmented output matrix. Note that the number of additional states corresponds to γ (i.e., the number of processing resources).

The discrete-time augmented matrices are defined by:

$$\begin{aligned} \Phi_d &= \begin{bmatrix} A_d & B_d & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \\ \Gamma_d &= \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}^T \\ C_d &= \begin{bmatrix} C_c & 0 & 0 & \dots & 0 & 0 \end{bmatrix}, \end{aligned} \tag{2.11}$$

where 0 denotes zero matrices of appropriate dimensions. The discrete-time pipelined model defined in Eq. 2.10 is used in the next section to design a controller with optimized QoC.

Example 2.2. ***Modelling pipelined system:*** *Consider the motivational example of Section 2.2. The model of the conveyor belt has two states $x(t) = \left[x^1(t)\ x^2(t)\right]^T$, one input $u(t)$, one output $y(t) = x^1(t)$, and the following matrices:*

$$A_c = \begin{bmatrix} 0 & 1.7 \\ -9 & -2.5 \end{bmatrix}, B_c = \begin{bmatrix} 0 \\ 10 \end{bmatrix}, C_c = [1\ 0].$$

A camera and an image-processing algorithm are used as a sensor for measuring the states $x^1(t)$ and $x^2(t)$, which correspond to the block positions and velocity, respectively. The total sensing-to-actuating delay is $\tau = 84$ ms. The open-loop step response has a rise time of $R_t = 337$ ms and a settling time of $S_t = 2.80$ s. Defining the sampling period using the rule of thumb of Eq. 2.3, results in $h_{Rt} = 33$ ms. However, the sensing-to-actuating delay forces computing the sampling period using Eq. 2.4, which results in $h_s = 84$ ms. Since $h_s > h_{Rt}$ the control performance is potentially limited.

An embedded platform with two processing resources is therefore used for the DISC, i.e., $\gamma = 2$. Using Eq. 2.5, the sampling period is then defined as $h = 42$ ms. The augmented state vector defined in Eq. 2.9 is given by: $z_k = \left[x_k^1\ x_k^2\ u_{k-2}\ u_{k-1}\right]^T$. Applying the discretization of Eq. 2.8 and the model augmentation of Eq. 2.11 gives:

$$\Phi_d = \begin{bmatrix} 0.986 & 0.070 & 0.015 & 0 \\ -0.358 & 0.886 & 0.397 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \Gamma_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, C_d = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T.$$

2.3.6 Alternative discretization of the pipelined model

We present an alternative model discretization, which captures the sensing-to-actuating delay by delaying the observation of the state vector rather than delaying the control input.

2.3.6.1 Model discretization

An alternative discretization to the model presented in Eq. 2.1 results in:

$$x_{k+1}^A = A_d x_k^A + B_d u_k^A, \tag{2.12}$$

where A_d and B_d are defined in Eq. 2.8, $x_k^A \in \mathbb{R}^{n\times 1}$ is the state vector with $x_k^A = x(kh)$, $k \in \mathbb{Z}^{\geq}$, and $u_k^A \in \mathbb{R}$ is the control input which is kept constant between sampling instants using a ZOH. This results in $u(t) := u_k^A$ for all $t \in [kh, (k+1)h)$. To capture the sensing-to-actuating delay, the control law may depend only on available (i.e., delayed) measured state vector:

$$u_k^A = f(x_{k-\gamma}^A). \tag{2.13}$$

Controllers with such a feedback impose restrictions on the controllability and design due to the fact that only an old delayed state vector is available for control computation. Moreover, common design strategies such as LQR and pole placement are not directly applicable [50].

2.3.6.2 Augmented non-delayed model

To obtain a canonical form of Eq. 2.12, the following state augmentation that keeps track of all relevant old state information is proposed:

$$z_k^A = \begin{bmatrix} (x_k^A)^T & (x_{k-1}^A)^T & \cdots & (x_{k-\gamma+1}^A)^T & (x_{k-\gamma}^A)^T \end{bmatrix}^T, \tag{2.14}$$

which results in the alternative discrete-time model of pipelined systems:

$$\begin{aligned} z_{k+1}^A &= \Phi_d^A z_k^A + \Gamma_d^A u_k^A \\ y_k^A &= C_d^A z_k^A, \end{aligned} \tag{2.15}$$

with $z_k^A \in \mathbb{R}^{n\gamma}$ the alternative augmented state vector, $u_k^A \in \mathbb{R}$ the alternative input vector, $y_k^A \in \mathbb{R}$ the alternative output vector, $\Phi_d^A \in \mathbb{R}^{(n\gamma)\times(n\gamma)}$ and $\Gamma_d^A \in \mathbb{R}^{n\gamma\times 1}$ the alternative augmented discrete-time state and input matrices, respectively, and $C_d^A \in \mathbb{R}^{1\times(n\gamma)}$ the alternative discrete-time augmented output matrix.

The alternative discrete-time augmented matrices are defined as:

$$\begin{aligned} \Phi_d^A &= \begin{bmatrix} A_d & 0 & \dots & 0 & 0 \\ I & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \dots \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & I & 0 \end{bmatrix} \\ \Gamma_d^A &= \begin{bmatrix} B_d^T & 0 & \dots & 0 & 0 \end{bmatrix}^T \\ C_d^A &= \begin{bmatrix} C_c & 0 & \dots & 0 & 0 \end{bmatrix}, \end{aligned} \tag{2.16}$$

where I and 0 denote identity and zero matrices of appropriate dimensions, respectively.

Compared to the model of Eq. 2.7, the model of Eq. 2.13 captures the sensing-to-actuating delay of a pipelined system by delaying the state-vector measurements rather than delaying the control input. Although delaying the state-vector measurements is a closer representation of how the real DISC works, it adds extra complexity to the control design due to the dependence of the control input on a delayed state vector. Moreover, the model of Eq. 2.13 stores old state vectors in the augmented state matrix instead of old controller inputs. This results in larger augmented matrices when the number of states is larger than one (i.e., $n > 1$), which is a common scenario in control systems. Larger augmented matrices result in an even more challenging control design, because it results in extra parameters for which tuning is required. Consequently, the modelling approach of Section 2.3.5 is preferred over the approach of Section 2.3.6.

2.4 State-of-the-art LQR tuning

In this section, we present the basics of LQR control and the state-of-the-art LQR tuning method based on classical PSO. Recall from Section 2.1 that we want to find a controller that minimizes the settling time for each resource configuration.

2.4.1 LQR controller

2.4.1.1 Control law

Consider the following control law:

$$u_k = Kz_k + Fr, \tag{2.17}$$

with $K \in \mathbb{R}^{1\times(n+\gamma)}$ the feedback control gain, $F \in \mathbb{R}$ the static feed-forward gain, and $r \in \mathbb{R}$ the reference. We apply $u_k = 0$ for $k < \gamma$ since we need to wait for the first sample to compute the control input. We consider a set-point regulation problem where the objective is to make sure $y_k \to r$ as $k \to \infty$.

2.4.1.2 Design objectives

The control law of Eq. 2.17 is designed to maximize a QoC metric defined by

$$QoC = \frac{1}{S_t},$$

with $S_t \in \mathbb{R}^+$ the controller settling time in seconds. The design objective is therefore to minimize settling time.

The settling time is determined simulating the response of the continuous-time system when the closed-loop discrete-time controller is used. To compute the settling time, the 2% criterion is used, i.e., the system needs to reach and remain within a bound of 2% around the reference r. The settling time is therefore computed as the time elapsed between a reference change and the system reaching (and remaining within) the aforementioned bound.

2.4.1.3 Feedback gain design

An LQR controller finds K minimizing the following cost function:

$$J = \sum_{k=0}^{\infty} \left(z_k^T Q z_k + u_k^T R u_k \right), \tag{2.18}$$

with $Q \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ and $R \in \mathbb{R}^{1\times 1}$ the state and input weight matrices, respectively, which are positive semi-definite and positive definite matrices, respectively, i.e., $Q = Q^T \geq 0$ and $R = R^T \succ 0$. These are the tuning parameters of the LQR [80, 58]. Note that $Q \geq 0$ does not guarantee that the resulting controller is always stable because $Q = 0$ might neglect the effect of the state vector in the cost function. To guarantee stability, an additional condition requires the pair (Q, Φ_d) to be observable (or detectable) [80]. Taking into account that this thesis deals with single-input single-output systems, R is a scalar value. Thus, we need that $R > 0$. We take into account these conditions during the controller design based on PSO.

The solution of Eq. 2.18 yields a non-linear equation known as the Ricatti Equation. Since there is no closed-form solution for this equation, iterative algorithms are commonly used to solve it [11]. To obtain an optimal controller for a time-domain metric such as settling time, the values of Q and R must be tuned. No analytic method exists to tune such matrices for time-domain metrics. In common approaches for tuning LQR (except [52]), Q and R are assumed to be diagonal. Consequently, each state and input value is weighted by one value in the tuning matrices reducing the design space. This results in an acceptable QoC and simple design procedure. However, by tuning all the elements of Q and R, a better QoC can be achieved.

2.4.1.4 Feed-forward gain design

Once K is found, a feed-forward gain F is calculated using Proposition 2.1 to achieve set-point regulation.

Proposition 2.1. *[58] The plant output y_k of the system of Eq. 2.10 with the control law of Eq. 2.17 converges to a constant reference r if the closed-loop system matrix*

$\Phi_{cl} = \Phi_d + \Gamma_d K$ is stable, the matrix $\begin{bmatrix} \Phi_d - I & \Gamma_d \\ C_d & 0 \end{bmatrix}$ is invertible, and F is defined by

$$F = \begin{bmatrix} K & I \end{bmatrix} \begin{bmatrix} \Phi_d - I & \Gamma_d \\ C_d & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

, with I and 0 *denoting identity and zero matrices of appropriate dimensions, respectively.*

Note that the control is designed in two steps i.e., first the feedback gain and then the feed-forward gain. This allows us to treat stability and performance aspects separately. Alternative design strategies such as Linear Quadratic Integrator (LQI) can also be used to design both gains in one step. In principle, these approaches can be adapted to our work. However, in the case of switched systems (which we deal with in Chapters 3 and 4), it is hard to find controllers that improve QoC in terms of time-domain metrics for LQI formulations [94]. This motivates us to use the current two-step approach.

Example 2.3. ***LQR tuning:*** *Consider the IBC of Example 2.2. The tuning of Q and R has the following form:*

$$Q = \begin{bmatrix} Q^{(1,1)} & Q^{(1,2)} & Q^{(1,3)} & Q^{(1,4)} \\ Q^{(1,2)} & Q^{(2,2)} & Q^{(2,3)} & Q^{(2,4)} \\ Q^{(1,3)} & Q^{(2,3)} & Q^{(3,3)} & Q^{(3,4)} \\ Q^{(1,4)} & Q^{(2,4)} & Q^{(3,4)} & Q^{(4,4)} \end{bmatrix}, R = \left[R^{(1,1)}\right],$$

where $Q^{(p,q)} \in \mathbb{R}$ with $p, q \in \{1,2,3,4\}$ and $R^{(1,1)} \in \mathbb{R}^+$ are constants to be tuned for the shortest settling time. Using Eq. 2.18, the cost reduces to:

$$J = \sum_{k=0}^{\infty} \big(x_k^1 Q^{(1,1)} x_k^1 + x_k^1 Q^{(1,2)} x_k^2 + x_k^1 Q^{(1,3)} u_{k-2} + x_k^1 Q^{(1,4)} u_{k-1} + \dots \\ + u_{k-1} Q^{(4,4)} u_{k-1} + u_k R^{(1,1)} u_k\big). \tag{2.19}$$

In case only the diagonal elements are tuned, the off-diagonal elements are set to zero (e.g., $Q^{(1,2)} = Q^{(1,3)} = Q^{(1,4)} = 0$, etc.). This means that the interaction between the states in the state vector is not considered in the minimization of the cost function.

We show that considering all the values of Q and R while tuning an LQR provides a better QoC in pipelined-sensing controllers.

2.4.2 PSO concepts

PSO is an evolutionary algorithm introduced by Kennedy and Eberhart in [70]. The algorithm was inspired by the behaviour of bird flocking and fish schooling, where a population of individuals collaboratively move using swarm intelligence to achieve a common goal such as finding the best feeding spot. In PSO, a population of individuals (particles) searches in parallel a problem design space to find a value that maximizes a fitness metric.

In PSO, the population is defined according to:

$$\mathbb{X}(l) = \left\{X_j(l) \mid j = 1,\ldots,m\right\},$$

with $X_j(l) \in \mathbb{R}^{f\times 1}$ a particle j in the population, $m \in \mathbb{Z}^+$ the population size (i.e., total number of particles), $l \in \mathbb{Z}^+$ the iteration number, and $f \in \mathbb{Z}^+$ the size of the design space. PSO has two phases: initialization and exploration. During the initialization phase, the particles in the population $\mathbb{X}$ are (commonly) randomly initialized. During the exploration phase, each individual is updated according to:

$$X_j(l+1) = X_j(l) + V_j(l), \tag{2.20}$$

with $V_j(l) \in \mathbb{R}^{f\times 1}$ the velocity of the particle j. The particles are updated until a stop criterion is met. Typical stopping criteria are the lack of progress for a fixed number of iterations or a maximum number of iterations reached.

The swarm intelligence of the PSO particles is achieved by the definition of the V_j. A common definition is:

$$\begin{split}V_j(l) = wV_j(l-1) + C_p rnd_1(l)\left(X_j(l) - Xpb_j(l)\right) + \\ C_g rnd_2(l)\left(X_j(l) - Xgb(l)\right),\end{split} \tag{2.21}$$

with $rnd_1(l)$, $rnd_2(l)$ random uniformly distributed numbers in the range $[0,1]$, $Xpb_j(l) \in \mathbb{R}$ is the particle j with the historically best fitness over all iterations, $Xgb(l) \in \mathbb{R}$ the particle with the historically global best fitness of the whole swarm and over all iterations, and C_p, C_g, and w tuning parameters. $Xpb_j(l)$ and $Xgb(l)$ are determined using a fitness metric $\mathscr{F}$:

$$Xpb_j(l) = \arg\max_{s=1,\ldots,l} \mathscr{F}\left(X_j(s)\right) \tag{2.22}$$

$$Xgb(l) = \arg\max_{\substack{s=1,\ldots,l,\\ j=1,\ldots,m}} \mathscr{F}\left(X_j(s)\right), \tag{2.23}$$

with $\mathscr{F}(X_j(s)) \in \mathbb{R}$ the fitness of the particle $X_j(s)$ at iteration s. The fitness metric corresponds to the optimization objective of the PSO algorithm.

PSO has four tuning parameters. The population size $m \in \mathbb{Z}^+$ is chosen according to the size of the design space [35]. The personal and global confidence $C_p \in \mathbb{R}^+$ and $C_g \in \mathbb{R}^+$ pull the speed towards $Xpb_j(l)$ and $Xgb(l)$, respectively. A small value for these parameters can limit the movement of the particles (implying they might fall into a local optimum) while a large value can cause the swarm to diverge. A typical value for these parameters is in the range of $[1,2]$ [54, 52, 35]. The inertia $w \in \mathbb{R}^+$ controls the exploration properties of the algorithm. A larger number yields a more global exploration while a smaller value results in a more local behaviour [35]. Typical values for w range in $[0,1]$ [54, 52, 127]. Note that Eq. 2.21 has three elements: one relative to the inertia that models the tendency of the particle to keep its previous direction, one relative to the personal element that models the attraction of the particle towards the best element it has ever found, and one relative to the attraction of the particle towards the best position ever found in the swarm. The tuning parameters are used to balance these elements.

PSO has advantages over other evolutionary algorithms such as genetic algorithms because of the fewer parameters to tune, less computational effort, and the use of swarm intelligence to evolve ($Xpb_j(l)$ and $Xgb(l)$) rather than a competition between particles (natural selection), which allows for a faster convergence [30].

2.4.3 Classical LQR tuning using PSO

For benchmarking purposes, the classical LQR-tuning PSO algorithm is presented in this section. Additionally, we update this algorithm to meet the stability conditions and the design objectives of Section 2.4.1. Recall that PSO has been used in a wide range of applications including tuning of LQR [35, 101, 127]. Classical approaches for tuning LQR using PSO assume that only the diagonal elements of Q and R are tuned (except for [52]).

2.4.3.1 Fitness metric

The fitness of the PSO corresponds to the desired QoC of the controller. The fitness selection varies from one application to the other. It can be selected as the cost function of the LQR [101], a modified cost function [35], the closed-loop poles location [127], or a time-domain criterion (e.g., settling time, steady-state error, rise time, and overshoot) [55]. Cost-related fitnesses are advantageous due to their ease of implementation because the cost is computed as per Eq. 2.18. However, they cannot be used in our application because when comparing different processing configurations the number of states changes (see Eq. 2.9). Therefore, the cost is affected by different states resulting in an unfair comparison. Determining the closed-loop pole locations requires more computational effort, but special properties of the resulting controller can be guaranteed (e.g., robustness). Robustness is a highly application-dependent QoC metric; therefore such a fitness is not useful either. Time-domain criteria can shape the controller response at a cost of extra computational effort caused by the fact that the plant's step response needs to be computed for every fitness evaluation. PSO is applicable to any of these fitness options.

In line with the design objectives of Section 2.4.1, the fitness is computed as:

$$\mathscr{F}\left(X_j(l)\right) = -S_{t,j}(l), \tag{2.24}$$

where $S_{t,j}(l) \in \mathbb{R}^+$ is the closed-loop settling time of the continuous-time system with the controller generated by the particle j at iteration l. The negation is to take into consideration the minimization problem of settling time.

2.4.3.2 Algorithm flow

A pseudo code of the classical PSO algorithm for tuning an LQR is shown in Algorithm 1. The algorithm starts generating a random population $\mathbb{X}$. Each particle in the population is defined according to:

$$X_j(l) = \left(Q_j^{(1,1)}(l), Q_j^{(2,2)}(l), \ldots, Q_j^{(\gamma+n,\gamma+n)}(l), R_j^{(1,1)}(l)\right),$$

where $Q_j^{(1,1)}(l), \ldots, Q_j^{(\gamma+n,\gamma+n)}(l)$ and $R^{(1,1)}$ are the (non-negative) diagonal elements of the Q_j and R_j matrices of particle j at iteration l. Note that the size of the design space corresponds to the number of diagonal elements in Q and R, i.e., $f = n + \gamma + 1$.

For each particle, the algorithm assembles the $Q_j(l)$ and $R_j(l)$ LQR tuning matrices based on the diagonal elements stored in $X_j(l)$. The observability condition of the pair $(Q_j(l), \Phi_d)$ is then checked. In case this condition is met, the controller gains and the fitness are computed. The fitness requires the computation of the controller settling time. This is done simulating the continuous-time system response when the discrete-time controller is used, as described in the design objectives of Section 2.4.1. The particle's personal best is then updated, if improved by the newly computed fitness. In case the observability condition is not met, the settling time of the particle is infinite, resulting in a fitness of minus infinity, corresponding to an unstable controller.

Once the finesses of each particle is computed, the global best of the swarm is updated, if it is improved by the newly computed fitness.

The algorithm proceeds then to update each particle according to the velocity update. To guarantee $Q_j(l+1) \geq 0$ and $R_j(l+1) > 0$, a constraint-violation correction is

Algorithm 1 Classical PSO algorithm

```
1: Input: system model, sampling period, PSO tuning parameters
2: Initialize $\mathbb{X}$
3: for $l = 1,\dots,$max iterations do
4:     for $j = 1,\dots,m$ do
5:         Create diagonal $R_j(l)$ and $Q_j(l)$ from $X_j(l)$
6:         if $(Q_j(l), \Phi_d)$ observable then
7:             Compute controller gains $K_j(l)$ and $F_j(l)$ (Eq. 2.18 and Proposition 2.1)
8:             Find fitness $\mathscr{F}\left(X_j(l)\right)$ (Eq. 2.24)
9:             Update $Xpb_j(l)$ (Eq. 2.22)
10:        else
11:            $\mathscr{F}(X_j(l)) = -\infty$
12:        end if
13:    end for
14:    Update $Xgb(l)$ (Eq. 2.23)
15:    for $j = 1,\dots,m$ do
16:        Find $V_j(l)$ (Eq. 2.21)
17:        Update particle to $X_j(l+1)$ (Eq. 2.20)
18:        Constraint-violation correction $(Q_j(l+1) \succeq 0, R_j(l+1) \succ 0)$
19:    end for
20:    if Stop criterion met then
21:        Stop algorithm
22:    end if
23: end for
24: Output: $Q$,$R$, and $K$ from $Xgb(l)$
```

implemented. Therefore, all the elements in each particle are kept non-negative and positive for $Q_j(l+1)$ and $R_j(l+1)$, respectively. Recall that a diagonal matrix with only non-negative entries results in positive semi-definiteness. Therefore, if any of the elements in the $X_j(l+1)$ violates the assumptions, then velocity $V_j(l)$ is successively halved until the resulting elements in $X_j(l+1)$ satisfy the assumptions.

The algorithm finishes when a stopping criterion is met, such as the criterion that no improvement in the global best is found for a fixed number of iterations or if a maximum number of iterations is reached.

2.5 Proposed LQR tuning

In this section, we present our method to tune all elements of Q and R, while guaranteeing $Q = Q^T \succeq 0$ and $R = R^T \succ 0$. In terms of the example presented in Example 2.3, we consider in the cost function all elements of the Q matrix including the off-diagonal elements.

2.5.1 Population definition

The classical PSO algorithm guarantees $Q_j(l) \succeq 0$ and $R_l(j) \succ 0$ by making these tuning parameters diagonal matrices with only non-negative and positive elements, respectively. In the proposed PSO, this is not directly applicable to $Q_j(l)$ because while exploring all its elements, negative definiteness can appear even if all the elements of $Q_j(l)$ are positive. Therefore, we use Proposition 2.2 below to define $Q_j(l)$. This proposition allows to create a positive semi-definite matrix $Q_j(l)$ through a random matrix $\hat{Q}_j(l)$. The PSO algorithm freely explores all the elements in $\hat{Q}_j(l)$, which will always lead to $Q_j(l) \succeq 0$. Guaranteeing $R_l(j) \succ 0$ is trivial for a single-input single-output system, which is the case for this thesis. However, to make the algorithm applicable to multiple-input multiple-output systems, one could follow an approach identical to $Q_j(l)$ for defining $R_j(l)$, taking into account Remark 2.1.

Each particle is then defined as the concatenation of all elements of the intermediate matrices $\hat{Q}_j(l)$ and $\hat{R}_j(l)$:

$$X_j(l) = \left(\hat{Q}_j^{(1,1)}(l), \hat{Q}_j^{(1,2)}(l), \ldots, \hat{Q}_j^{(1,\gamma+n)}(l), \hat{Q}_j^{(2,1)}(l), \hat{Q}_j^{(2,2)}(l), \ldots, \hat{Q}_j^{(n+\gamma,n+\gamma)}(l), \hat{R}_j^{(1,1)}(l)\right)$$

with $\hat{Q}_j^{(1,1)}(l), \ldots, \hat{Q}_j^{(n+\gamma,n+\gamma)}(l)$ all the elements of the matrix $\hat{Q}_j(l)$, $\hat{Q}_j(l) \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ an intermediate matrix to compute $Q_j(l)$, $\hat{R}_j^{(1,1)}(l)$ the element of the matrix $\hat{R}_j(l)$, $\hat{R}_j(l) \in \mathbb{R}^{1\times1}$ and intermediate matrix to compute $R_j(l)$ according to:

$$R_j^{(1,1)}(l) = (\hat{R}_j^{(1,1)}(l))^2. \tag{2.25}$$

In this case, the design space of the PSO covers all the elements in the Q and R matrices, i.e., the size of the design space is $f = (n+\gamma)^2 + 1$.

Proposition 2.2. *Let Q be a symmetric matrix of order i. Q is positive semi-definite if there exists a matrix $\hat{Q} \in \mathbb{R}^{i\times i}$ of rank$(\hat{Q}) \leq i$, such that $Q = \hat{Q}^T \hat{Q}$.*

Proof. [32, Proposition 2.62]. □

Remark 2.1. *Note that if in Proposition 2.2, rank($\hat{Q}$) = i, then Q would be positive definite.*

Example 2.4. ***PSO particle definition:*** *Continuing with Example 2.3, the j^{th} particle is defined as:*

$$X_j(l) = \left(\hat{Q}_j^{(1,1)}(l), \hat{Q}_j^{(1,2)}(l), \hat{Q}_j^{(1,3)}(l), \hat{Q}_j^{(1,4)}(l), \hat{Q}_j^{(2,1)}(l), \ldots, \hat{Q}_j^{(4,4)}(l), \hat{R}_j^{(1,1)}(l)\right),$$

where

$$\hat{Q}_j(l) = \begin{bmatrix} \hat{Q}_j^{(1,1)}(l) & \hat{Q}_j^{(1,2)}(l) & \hat{Q}_j^{(1,3)}(l) & \hat{Q}_j^{(1,4)}(l) \\ \hat{Q}_j^{(2,1)}(l) & \hat{Q}_j^{(2,2)}(l) & \hat{Q}_j^{(2,3)}(l) & \hat{Q}_j^{(2,4)}(l) \\ \hat{Q}_j^{(3,1)}(l) & \hat{Q}_j^{(3,2)}(l) & \hat{Q}_j^{(3,3)}(l) & \hat{Q}_j^{(3,4)}(l) \\ \hat{Q}_j^{(4,1)}(l) & \hat{Q}_j^{(4,2)}(l) & \hat{Q}_j^{(4,3)}(l) & \hat{Q}_j^{(4,4)}(l) \end{bmatrix}.$$

2.5.2 LQR tuning for a given number of pipes

Our algorithm is shown in Algorithm 2. The algorithm is divided into two phases: initialization and exploration. During the initialization phase, the population size is chosen according to the number of variables to tune. By means of simulations, we propose the following initial values for the tuning parameters. For the population size m, we take

$$2\gamma m_u + e^{m_c \gamma}, \tag{2.26}$$

rounded to the nearest natural number, where $m_u = (n+1)^2 + 1$ is the number of unknown values that the PSO needs to find (i.e., the total number of elements in Q and R for the augmented model) with one sensing pipe. m_c is a design parameter that depends on the complexity of the dynamic model. The exponential term increases the swarm size with the number of pipes. This is necessary because the size of the design space f also grows with the number of pipes according to $f = (n+\gamma)^2 + 1$. The population can be initialized with random values in the set of real numbers. During the exploration phase, a positive semi-definite $Q_j(l)$ and a positive $R_j(l)$ are calculated according to Proposition 2.2 and Eq. 2.25, respectively. If the pair $(Q_j(l),\ \Phi_d)$ is observable, the controller gains, the fitness, and the personal and global best are found. Note that the rank condition from Proposition 2.2 is not explicitly checked because the observability condition suffices to guarantee that the resulting controller is stable. If the pair $(Q_j(l),$ $\Phi_d)$ is not observable (because, for example $Q_j(l) = 0$), then the settling time of the particle is set to infinite. The iterations are stopped when the global best is unchanged during a fixed number of iterations or the maximum number of iterations is reached. With this method, the settling time is minimized for a fixed number of pipes by tuning all values of Q and R.

Algorithm 2 Proposed PSO algorithm

```
 1: Input: system model, sampling period, PSO tuning parameters
 2: Initialize $\mathbb{X}$
 3: for $l = 1,\ldots,$ max iterations do
 4:     for $j = 1,\ldots,m$ do
 5:         Create $\hat{Q}_j(l)$, $\hat{R}_j(l)$ from $X_j(l)$
 6:         Find $R_j(l)$ (Eq. 2.25) and $Q_j(l)$ (Proposition 2.2)
 7:         if $(Q_j(l),\Phi_d)$ observable then
 8:             Compute controller gains $K_j(l)$ and $F_j(l)$ (Eq. 2.18 and Proposition 2.1)
 9:             Find fitness $\mathscr{F}\left(X_j(l)\right)$ (Eq. 2.24)
10:             Update $Xpb_j(l)$ (Eq. 2.22)
11:         else
12:             $\mathscr{F}\left(X_j(l)\right) = -\infty$
13:         end if
14:     end for
15:     Update $Xgb(l)$ (Eq. 2.23)
16:     for $j = 1,\ldots,m$ do
17:         Find $V_j(l)$ (Eq. 2.21)
18:         Update particle $X_j(l+1)$ (Eq. 2.20)
19:     end for
20:     if Stop criterion met then
21:         Stop algorithm
22:     end if
23: end for
24: Output: $Q$,$R$,$K$ from $Xgb(l)$
```

Example 2.5. ***PSO tuning results:*** *Continuing with Example 2.4, the PSO algorithm is implemented using Matlab. The population size is defined with* $m_c = 0.7$, $\gamma = 2$, *and* $m_u = 10$, *which results in a swarm with* $m = 44$ *particles. Each particle is initialized with random numbers in the range* $[-1,1]$. *The tuning parameters were experimentally chosen as:*

$$w = 0.5, C_p = 1.5, C_g = 1.5. \tag{2.27}$$

As stopping criterion, we used 200 *as a maximum number of iterations and 10 for a fixed number of iterations with no change in the global best. Via numeric experimentation, we established that this number of iterations are upper margins that guarantee no other global best is found by the optimization algorithm. The controller finds an* *optimal settling time of $S_t = 123.5$ ms. The resulting controller gains are:*

$$K = \begin{bmatrix} -30.9 & -7.3 & -2.7 & -1.9 \end{bmatrix}, F = 36.1.$$

2.6 LQR tuning: comparison

The resulting settling time of the tuned LQR controllers for our running example with different numbers of pipes is shown in Fig. 2.5. We compared our method with two benchmarking strategies for designing controllers: Bryson's method [20] and the classical PSO for tuning the diagonal of Section 2.4.3. Bryson's method results in a stable controller, although it is not meant to produce minimum settling time. Both benchmarking strategies are implemented using Matlab.

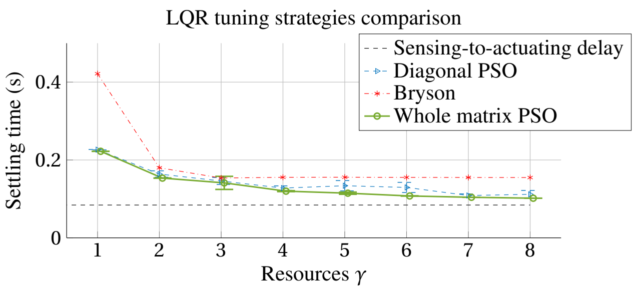


Figure 2.5: Comparison of LQR tuned with different strategies on our motivational example.

With Bryson's method, the maximum desirable deviation of the states and controller input is used to find an initial value of the diagonal of Q and R. Eq. 2.18 is minimized resulting in a feedback gain that is used to compute the controller step response. If the resulting settling time is unsatisfactory, Q and R are manually adjusted and the process is repeated.

The parameters of the PSO algorithm for tuning the diagonal are the same as for the PSO tuning the whole matrices, i.e., Eq. 2.21 is used to update the swarm, Eq. 2.24 is used as fitness, Eq. 2.26 is used to select the population size with $m_c = 0.7$, and w, C_p, and C_g are as considered in Eq. 2.27. Parameter m_u is adjusted according to the number of free variables with one pipe, i.e., $m_u = 10$ for the proposed PSO algorithm while $m_u = 4$ for the classical PSO algorithm.

The random initialization of the PSO algorithm might produce different results between equal executions. Hence, each simulation is executed 10 times for each number of pipes. We plot the average settling time of such consecutive runs and we include an error bar which corresponds to the standard deviation in Fig. 2.5. A large standard deviation means that the PSO algorithms are converging to different local optima in every algorithm repetition. This happens because the population is not large enough to explore the whole design space and it is solved by enlarging the population. The number of pipes is increased till the QoC improvement is negligible.

Fig. 2.5 shows that the controllers tuned using Bryson's method initially improve the settling time with each additional resource added. However, such an improvement stops with three pipes, because the design space is too large, leading to controllers being incorrectly tuned. Tuning the diagonal of Q and R using PSO outperforms the settling time found by the Bryson method because of the more extensive search in the design space. A trade-off analysis with this method suggests that the QoC of the controller is improved with every new pipe, except when using five and six pipes.

Tuning the whole matrix performs equal to or better than tuning only the diagonal. The result of a trade-off analysis with this method differs from the previous cases. The settling time of the controller is always improved with each newly added pipe. However, after a certain point adding more pipes generates a minimal improvement in the settling time at a cost of extra processing resources. Our method is therefore more suitable to analyse the trade-off to decide in a design phase how many pipes are meaningful to implement in a DISC, when compared to the benchmarked strategies. Note that the sensing-to-actuating delay denotes an asymptote for the settling time of all the controllers, because the settling time cannot be shorter than the delay.

2.7 Trade-off analysis

This section explores the trade-off between processing resources and QoC (i.e. settling time). We show that the amount of sensing-to-actuating delay is a critical factor in such a trade-off. The simulations are based on our method described in Section 2.5 and the motivational example from Example 2.2 and following. We apply the proposed LQR tuning method for different processing configurations. Then we analyse the resulting settling time and we present some observations.

2.7.1 Processing resources vs settling time

Fig. 2.3 shows that the use of more sensing pipes allows to shorten the settling time. This creates a processing resources - settling time trade-off. To analyse such a trade-off, an initial resource configuration of one pipe is considered. Our method is used to tune an LQR with the minimum settling time. The processing resources are increased and the process is repeated until the improvement in settling time is negligible. The result is shown in Fig. 2.6. Note that for the case of three pipes a large standard deviation is observed. This is because the PSO algorithm is converging to different local optima in every algorithm repetition, which can be solved by increasing the population size.

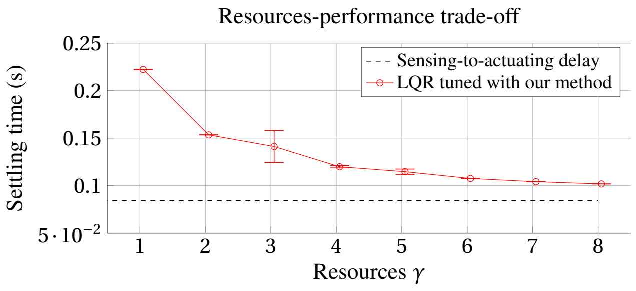


Figure 2.6: Trade-off between settling time and processing resources with fixed delay of $\tau = 0.084$.

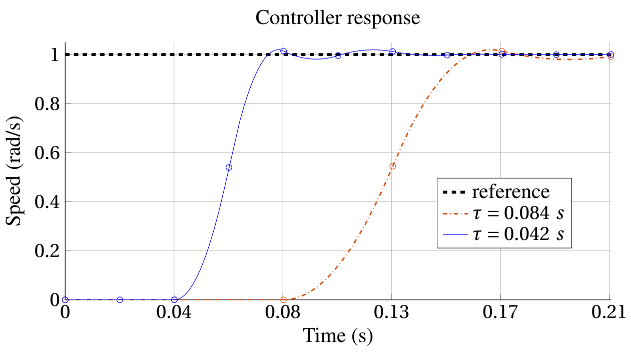
2.7.2 Settling time vs delay

The trade-off between processing resources and settling time depends not only on the model dynamics but also on the amount of sensing-to-actuating delay. This is better illustrated in Fig. 2.7, where a two-core LQR controller is tuned considering two different delays. The settling time (with the delay subtracted) is 0.035 s for $\tau = 0.042$ s and 0.069 s for $\tau = 0.084$ s (See Fig. 2.7b). Note that the sampling period is chosen according to Eq. 2.5, hence different delays give different h. The difference in the settling time in Fig. 2.7 is due to the difference in sampling period.

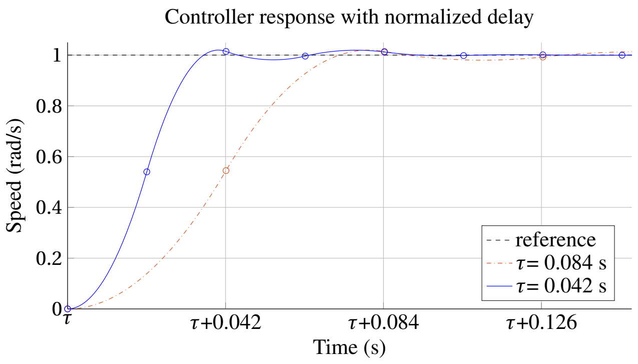
We illustrate this design consideration by varying the sensing-to-actuating delay in the motivational example. The effect of the delay on the controller settling time is shown in Fig. 2.8 and Fig. 2.9. Fig. 2.8 shows the effect of a wide range of delays on the settling time. Two sensing cores are considered for this simulation. In Fig. 2.9 the settling time of models with two delays is compared over a range of sampling periods. The processing resources are varied in order to change the sampling period. In both figures, the delay is subtracted from the settling time for comparison purposes.

2.7.3 Observations

We observed the following from our simulations:



(a) Plant responses.



(b) Plant responses with delay subtracted in the horizontal axis.

Figure 2.7: Control responses of motivational example with two sensing cores but different delays τ.

- Fig. 2.6 shows that the settling time gets shorter with each newly added pipe. However, the improvement does not grow proportionally with the number of pipes. For example, after four pipes the settling time improvement is reduced to an order of milliseconds. Therefore, there is little benefit from the extra processing resources. Clearly, for the plant under consideration and the given sensing-to-actuating delay, a higher number of pipes is (practically) beneficial until four pipes. Obviously, such region of interest depends on the plant dynamics and the sensing-to-actuating delay. This should be considered in the design phase of a pipelined controller in general to achieve a better QoC / resource trade-off.

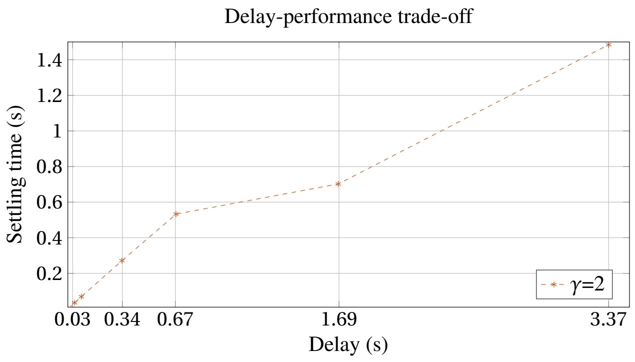


Figure 2.8: Impact of sensing-to-actuating delay on the settling time (with delay subtracted for comparison purposes) using two sensing cores as processing resources.

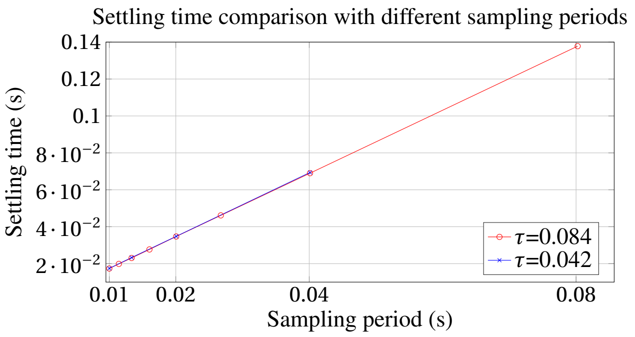


Figure 2.9: Impact of sampling period on settling time (with sensing-to-actuating delay subtracted for comparison purposes).

- Fig. 2.8 shows that the improvement of settling time depends on the sensing-to-actuating delay. The shortest settling time is achieved with the shortest delay, because the resulting sampling period becomes similar to the value recommended by Eq. 2.3. When the resulting sampling period is larger, the settling time gets longer significantly because the model becomes under-sampled. Therefore, given a sensing-to-actuating delay, the number of sensing pipes should be chosen to make sure that the sampling period is equal or smaller to the recommendation of Eq. 2.3.

- Fig. 2.9 shows the relationship between sampling period and settling time. The settling time gets longer proportionally with the sampling period. Note that if different delays have the same sampling period due to differences in processing resources (e.g., in Eq. 2.5 $\tau = 0.084\ s$ with $\gamma = 2$ produces $h = 0.042\ s$ and $\tau = 0.042\ s$ with $\gamma = 1$ produces $h = 0.042\ s$) the resulting settling time is identical. Therefore, for this example the settling time (with delay subtracted) achieved with a short delay can also be achieved with a long delay, if sufficient processing resources are added. This is useful for applications where shortening the sensing-to-actuating delay is not feasible and subsequent samples can be processed independently.

2.8 Summary

In this chapter, we have presented a method to analyse the trade-off between processing resources and settling time in pipelined-sensing control, which shows the impact of processing resources on control performance. The method uses PSO for finding all elements in the tuning matrices of an LQR that minimize the settling time. Next, the settling time is investigated with varying processing resources and sensing-to-actuating delay.

For a motivational example, the trade-off analysis shows that each newly added processing resource shortens the settling time. However, after a certain point the improvement becomes negligible. We also showed that the improvement in settling time is affected by the amount of delay: longer delay produces longer sampling periods which in turn produces a longer settling time. Finally, we showed that the same model with two different sensing-to-actuating delays can obtain the same settling time, if their processing resources lead to the same sampling period.

The results presented in this chapter are obtained using one control-design technique (i.e., LQR), which suffices to show the impact of processing resources in control performance. The results can be straightforwardly extended to other control-design techniques. For example, adding an integral action to the LQR can allow the controller to reject a wider range of disturbances (as is done in Chapter 5). Alternative control-design strategies (e.g., explicitly considering the sensing-to-actuating delay into the control design or using multi-rate strategies where sensing-to-actuating delay is longer than the actuation period) can also be applied to pipelined control. However, to apply these alternative techniques, additional modelling or control design strategies might be required. This remains as an interesting research opportunity.

The method presented in this chapter builds the basis of control design in the next chapter. The method does not consider the presence of the model inaccuracy, which might be present in real-life plants. In the next chapter, we extend the trade-off analysis to take model inaccuracy into account.

Chapter 3

Resources vs QoC trade-off analysis under model uncertainties

3.1 Problem formulation 44
3.2 Related work 45
3.3 Modelling and control design of a nominal system 46
3.4 Overview of the proposed design flow 49
3.5 Modelling pipelined systems with uncertainties 51
3.6 Model-uncertainty approximation 54
3.7 Robustness of pipelined systems 57
3.8 Robustness example of pipelined systems 62
3.9 Feasibility study 65
3.10 Summary 74

Pipelining the sensing algorithm improves QoC with each newly added processing resource. However, after a certain point the QoC improvement becomes negligible. This creates a trade-off between resource usage (i.e., cost of implementation) and QoC. We introduced a method to analyse such a trade-off in Chapter 2, which assumes perfect knowledge of the plant model. However, when designing control systems it is common to encounter uncertainties in the model parameters, which may have a negative impact on the QoC. In pipelined control, the QoC is also deteriorated which may impact the aforementioned trade-off. In this chapter, we present an analysis framework to include model uncertainties in the afore-mentioned trade-off (between resource usage and QoC). We present a technique to approximate discrete-time uncertainties based on the continuous-time uncertainties for given uncertainty bounds. To approximate such uncertainty bounds for a real system, we consider uncertainties in one element of the system matrices. Uncertainties and their impact in the performance-oriented designs of Chapter 2 are studied. We also provide a robustness-oriented pipelined controller design that maximizes the tolerable uncertainties in the control loop. Our results show that in performance-oriented designs, the tolerable uncertainties for a pipelined controller decrease when increasing the number of pipes. In robustness-oriented designs, the controller robustness is enhanced with each newly added pipe. We show the feasibility of our technique by implementing a realistic example in a Hardware-In-the-Loop simulation.

The contents of this chapter were published in [92].

3.1 Problem formulation

Uncertainties in the plant model are a well-known factor of QoC deterioration, which can lead to control instability [148, Chapter 8.1]. In pipelined-sensing control, such a QoC deterioration affects the trade-off between processing resources and QoC. For example, in the motivational example of Chapter 2, the trade-off analysis indicated that four pipes was the number of cores that still gives a meaningful improvement in QoC. However, if there is a maximum uncertainty of 2% in one of the elements of the state and input matrices (i.e., there is a robustness constraint that the controller continues to be stable even with up to 2% deviation in one element of these matrices), the controller with four pipes becomes infeasible (i.e., the stability is no longer guaranteed) whereas the one with three pipes remains stable for the same uncertainties, while it still gives a meaningful improvement in QoC.

There is substantial literature dealing with the stability analysis of a controller based on a discrete-time model with delay and uncertainties (see for example [137, 47]). Time-invariant norm-bounded uncertainties are common in such analyses because in many physical systems the exact value of the parameters is not known but it is limited to a range of possible values (see for example [111, 18, 78, 19]). Analysing the effect of time-invariant bounded uncertainties has not been done yet for pipelined systems. Moreover, discrete-time robustness-analysis techniques commonly start from known discrete-time uncertainties (see for example [38]). The relation of such discrete-time uncertainties with real-life continuous-time applications is not obvious. In the trade-off analysis of pipelined systems, this relation is particularly relevant because each newly added pipe produces a different sampling time implying that new discrete-time uncertainties have to be computed. In other words, the same continuous-time uncertainties have different effects in pipelined-sensing controllers with different resource configurations. This affects the trade-off analysis.

The contributions of this chapter are threefold. (i) We approximate discrete-time uncertainties based on continuous-time time-invariant norm-bounded uncertainties described by matrices with a single non-zero element (i.e., a single uncertain element in each matrix). The resulting uncertainties are time-invariant and norm-bounded. (ii) We present a robustness-analysis technique that shows the impact of the approximated uncertainties in performance-oriented pipelined controllers and in the trade-off between resource usage and QoC. For that we adapt the technique of [137, 47] for pipelined systems. (iii) We present a robustness-oriented controller design that uses the approximated uncertainties to enhance robustness with each added pipe. Our contributions capture the interplay between processing resources, control performance, and control robustness in pipelined systems. We show the feasibility of our approach by implementing a pipelined controller on a platform with parallel processing capabilities using Hardware-In-the-Loop simulation.

3.2 Related work

Pipelined-sensing control has been used for systems with image-based sensing delay in [75, 76, 23, 90, 91] and for systems with network-based sensing delay in [144, 135, 136]. This literature focuses on comparing pipelined sensing with serial sensing (e.g., [75]), using frequency domain information to analyse the phase margin of different pipelined controllers [144], allocating resources in networked control to achieve pipeline parallelism e.g., [135], and introducing modelling (e.g., [90]) and control-design strategies (e.g., [91]). Analysing the QoC of pipelined control with respect to a model with uncertainties has not been reported before.

The impact of bounded uncertainties on control systems has been widely studied in the robust control literature for the continuous-time domain [73, 141, 140] and the discrete-time domain [38, 34, 150]. A pipelined-sensing control corresponds to a discrete-time controller; therefore we focus on discrete-time techniques. Discrete-time approaches commonly assume a known set of discrete-time uncertainties. In a pipelined-sensing control, the uncertainties originate from the dynamic system in the continuous-time domain, whereas their impact has to be analysed on the discrete-time pipelined controller. Therefore, a strategy to discretize a model with uncertainties is required. Strategies to compute discrete-time approximations of the uncertainty matrices are found in using the Chebyshev quadrature [111], switched systems [61] or first-order forward Euler approximations [78, 74, 131]. However, these strategies correspond to first and second order approximations which might not accurately represent the continuous-time system. In pipelined control, each resource configuration varies the sampling period generating different discrete-time uncertainties. The above approximations might not produce enough difference between resource configurations, making them not suitable for this application. Taylor series have been used in [18] to discretize continuous-time uncertainties using the first i terms of the series. The resulting model is a homogeneous polynomial of degree i. Increasing i in the series expansion leads to a higher accuracy in the discretization at the cost of a more complex discrete-time model. This technique might be applicable to pipelined control; however the resulting homogeneous polynomials are not commonly used in discrete-time robustness-analysis techniques. We present a technique to approximate bounded discrete-time uncertainties, that eases the robustness analysis.

The combined effect of delay and discrete-time bounded uncertainties on discrete-time controllers has been analysed following two approaches: predictor output [41, 85, 47] or static feedback [137, 47]. Predictor-output approaches use an estimate of the system output after the delay to compute the controller output. Static-feedback approaches include the delay in the discrete-time model to compute the controller output. The pipelined controller designed in [91] corresponds to a static feedback approach. Static-feedback literature with uncertainties has reported a control-design strategy in the presence of constant and variable time delay in [137] and an uncertainty-maximization technique with variable time-delay for a given controller in [47]. In this chapter, we apply the static-feedback technique to pipelined systems which have discretized time-invariant norm-bounded uncertainties. We analyse the effect of model uncertainties on the trade-off between processing resources and QoC in a performance-oriented design, while meeting a robustness constrain. We also develop a robustness-oriented pipelined controller design.

3.3 Modelling and control design of a nominal system

This section summarizes the pipelined control strategy of Chapter 2. To this end, we present an overview of the contents of Section 2.3 and Section 2.5 in the following sections.

3.3.1 Nominal model in pipelined control

This subsection summarizes the modelling strategy presented in Section 2.3 for pipelined-sensing systems without uncertainties (i.e., for nominal systems).

Given the continuous-time nominal plant:

$$\begin{aligned}\dot{x}(t) &= A_c x(t) + B_c u(t)\\ y(t) &= C_c x(t),\end{aligned} \tag{3.1}$$

with $x(t) \in \mathbb{R}^{n\times 1}$ the state vector, $u(t) \in \mathbb{R}$ the control input, $y \in \mathbb{R}$ the output, and $A_c \in \mathbb{R}^{n\times n}$, $B_c \in \mathbb{R}^{n\times 1}$ and $C_c \in \mathbb{R}^{1\times n}$ the nominal state, input and output matrices, respectively. $t \in \mathbb{R}^{\geq}$ is the time and $n \in \mathbb{Z}^{+}$ corresponds to the number of states.

Considering an embedded system with $\gamma \in \mathbb{Z}^{+}$ processing resources available for the DISC, the controller sampling period $h \in \mathbb{R}^{+}$ is defined by:

$$h = \frac{\tau}{\gamma}, \tag{3.2}$$

where $\tau \in \mathbb{R}^{+}$ is the sensing-to-actuating delay. This leads to a discrete-time pipelined model of the form (see Eq. 2.10):

$$\begin{aligned}z_{k+1} &= \Phi_d z_k + \Gamma_d u_k\\ y_k &= C_d z_k,\end{aligned} \tag{3.3}$$

where $k \in \mathbb{Z}^{\geq}$ is the discrete-time index, $z_k \in \mathbb{R}^{(n+\gamma)\times 1}$ is the augmented state vector further explained in Eq. 2.9, $\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$, $\Gamma_d \in \mathbb{R}^{(n+\gamma)\times 1}$ and $C_d \in \mathbb{R}^{1\times(n+\gamma)}$ are the nominal augmented discrete-time state, input and output matrices, respectively, and $u_k \in \mathbb{R}$ is the control input. The control input is implemented using a ZOH to keep the actuation signal constant between consecutive sampling periods. This results in $u(t) := u_{k-\gamma}$ for all $t \in [kh, (k+1)h)$.

The discrete-time input and state matrices are defined by:

$$\Phi_d = \begin{bmatrix} A_d & B_d & 0 & \dots & 0\\ 0 & 0 & 1 & \dots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \dots & 1\\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}, \Gamma_d = \begin{bmatrix} 0\\ 0\\ \vdots\\ 0\\ 1 \end{bmatrix}, C_d = \begin{bmatrix} C_c^T\\ 0\\ \vdots\\ 0\\ 0 \end{bmatrix}^T, \tag{3.4}$$

with $A_d \in \mathbb{R}^{n\times n}$, $B_d \in \mathbb{R}^{n\times 1}$, and $C_d \in \mathbb{R}^{1\times(n+\gamma)}$ the discretization of A_c, B_c, and C_c, respectively with sampling period h and Eq. 2.8.

The control law is defined by:

$$u_k = Kz_k + Fr, \tag{3.5}$$

where $K \in \mathbb{R}^{1\times(n+\gamma)}$ is the feedback gain, $F \in \mathbb{R}$ is the feed-forward gain, and $r \in \mathbb{R}$ is the reference. We apply $u_k = 0$ for $k < \gamma$ since we need to wait for the first sample to compute the control input. For the closed-loop representation, we define:

$$\Phi_{cl} = \Phi_d + \Gamma_d K, \tag{3.6}$$

with $\Phi_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ the discrete-time nominal closed-loop matrix, which is used in the robustness analysis of Section 3.7.

3.3.2 Control design and trade-off analysis

This subsection summarizes the method described in Section 2.5 for tuning a controller with pipelined sensing.

To analyse the trade-off between resource usage and QoC, a controller with optimized performance has to be tuned for each resource configuration. Performance metrics related to the response of the controller (e.g., tracking velocity, settling time) are of interest in many real-life applications, e.g., [75, 76]. In line with the arguments of Section 1.2, we consider settling time as a QoC metric. We then define our QoC performance as:

$$QoC = \frac{1}{S_t},$$

where $S_t \in \mathbb{R}^+$ is the controller settling time. The feedback gain K defined in Eq. 3.5 is designed using the well-known optimal design strategy LQR. In line with the reasoning of the previous chapter, LQR is used as a benchmarking strategy because it allows to extend the results from Chapter 2. However, note that other control strategies can be applied here. The LQR finds a K that minimizes the quadratic cost:

$$J = \sum_{k=0}^{\infty} \left(z_k^T Q z_k + u_k^T R u_k\right), \tag{3.7}$$

with $Q \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ the state weight matrix and $R \in \mathbb{R}^{1\times1}$ the input weight matrix of the LQR, which satisfy $Q = Q^T \geq 0$, $R = R^T > 0$, and with the pair (Q, Φ_d) observable [80]. The feed-forward gain F of Eq. 3.5 can then be designed according to the set-point regulation equation of Proposition 2.1. Finding F can be straightforwardly done once K is available. However, for finding K note that the cost function of Eq. 3.7 gives a controller which is optimal with respect to J in terms of the tuning parameters Q and

R; this does not necessarily mean that it has the shortest settling time. We refer to LQR tuning as the process of finding a Q and an R that minimize the settling time of the system.

PSO is used to tune an LQR for minimum settling time. A PSO algorithm employs a swarm of m particles X_j with $j = 1 \dots m$, to explore the design space of a problem in order to minimize a fitness metric. In this case, the design space corresponds to the values of Q and R for particle j, Q_j and R_j, and the fitness metric corresponds to the controller settling time. The challenge arises because Q_j has to remain positive semi-definite while it is explored by the PSO algorithm. To address this, the PSO algorithm presented in Chapter 2 uses two intermediate matrix variables $\hat{Q}_j$ and $\hat{R}_j$ to define each particle in the swarm i.e., $X_j = (\hat{Q}_j^{(1,1)}, \hat{Q}_j^{(1,2)}, \dots, \hat{Q}_j^{(n+\gamma,n+\gamma)}, \hat{R}_j^{(1,1)})$. The tuning parameters are then computed using $\hat{Q}_j$ and $\hat{R}_j$ with Proposition 2.2 and Eq. 2.25, respectively. Proposition 2.2 guarantees that the resulting Q_j is positive semi-definite. PSO can then be used to find the values of $\hat{Q}_j$ and $\hat{R}_j$ that minimize the settling time for a resource configuration. Further details about the proposed PSO can be found in Section 2.5.

The method discussed in this subsection is used to explore the trade-off between processing resources and QoC. However, in some applications, it is common to have modelling uncertainties, which combined with the pipelined delay potentially affect the controller stability and the trade-off analysis. It is then necessary to analyse the robustness of the designed controller. To do so, the design flow of the following section is proposed.

3.4 Overview of the proposed design flow

To analyse the robustness of a performance-oriented pipelined controller or to design a robustness-oriented controller, the following design flow is proposed, clarified in Fig. 3.1.

1. **Discretize model**: Given a continuous-time nominal dynamic model, a sensing-to-actuating delay, and a set of resource configurations (i.e., the number of sensing cores used), a discrete-time nominal pipelined model is found for each resource configuration (as outlined in Section 3.3.1).

2. **Performance-oriented controller design**: Given the discrete-time nominal pipelined models for each resource configuration, we use PSO to tune a controller with minimum settling time as an optimization objective (as outlined in Section 3.3.2). Note that the controller is designed using the nominal plant.

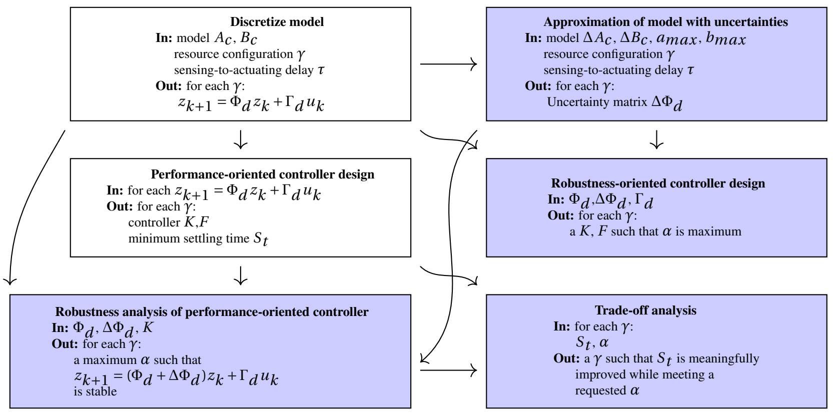

Discretize model
In: model A_c, B_c
resource configuration γ
sensing-to-actuating delay τ
Out: for each γ:
$z_{k+1} = \Phi_d z_k + \Gamma_d u_k$
Approximation of model with uncertainties
In: model $\Delta A_c, \Delta B_c, a_{max}, b_{max}$
resource configuration γ
sensing-to-actuating delay τ
Out: for each γ:
Uncertainty matrix $\Delta\Phi_d$
Performance-oriented controller design
In: for each $z_{k+1} = \Phi_d z_k + \Gamma_d u_k$
Out: for each γ:
controller K,F
minimum settling time S_t
Robustness-oriented controller design
In: $\Phi_d, \Delta\Phi_d, \Gamma_d$
Out: for each γ:
a K, F such that α is maximum
Robustness analysis of performance-oriented controller
In: $\Phi_d, \Delta\Phi_d, K$
Out: for each γ:
a maximum α such that
$z_{k+1} = (\Phi_d + \Delta\Phi_d)z_k + \Gamma_d u_k$
is stable
Trade-off analysis
In: for each γ:
S_t, α
Out: a γ such that S_t is meaningfully
improved while meeting a
requested α

Figure 3.1: Overview of the proposed design flow. Coloured blocks correspond to the contributions made in this chapter.

We subsequently analyse if the designed controller is robust against given uncertainties in Step (4).

3. **Approximation of model with uncertainties:** Given continuous-time uncertainties for the dynamic model, the set of resource configurations, and the sensing-to-actuating delay, a discrete-time model with uncertainties is found for each resource configuration. Section 3.5 presents a discrete-time pipelined model with uncertainties while Section 3.6 describes the discrete-time approximation of the model with uncertainties. The approximation is the basis for analysis (Steps (4) and (5)) or design (Step (6)). If a robustness-oriented pipelined controller is desired, Step (6) is implemented. If the robustness analysis of a performance-oriented control is desired, Steps (4) and (5) are applied.

4. **Analyse robustness of performance-oriented controller**: Given the discrete-time controller and the model with uncertainties, we analyse the robustness of each resource configuration. To do so, we parametrize the uncertainties as a scalar α, which is maximized for guaranteed stability. Section 3.7.1 presents basic control robustness theory while the details of the robustness analysis are presented in Section 3.7.2.

5. **Trade-off analysis**: Given the settling times and the robustness analysis (i.e., an α) for each resource configuration, a resource configuration is chosen such

that the settling time is still meaningfully improved while a robustness constraint (i.e., a minimum desired α) is met. Section 3.8 shows a detailed example of this analysis.

6. **Robustness-oriented controller design**: Given the discrete-time pipelined model with uncertainties, a controller is designed with maximum robustness for each resource configuration. To do so, we find controllers that maximize the parametrized scalar α. Details are given in Section 3.7.3.

3.5 Modelling pipelined systems with uncertainties

To perform a robustness analysis of the discrete-time pipelined controller, the model with uncertainties has to comply with a particular matrix structure in the discrete-time domain. Therefore, in this section, we describe the required structure of the discrete-time uncertainties and their relationship with their continuous-time counterparts. An approximation strategy for discretizing the uncertainties is next explained in Section 3.6.

3.5.1 Continuous-time model with uncertainties

Consider that the continuous-time system described in Eq. 3.1 contains uncertainties:

$$\begin{aligned} \dot{x}(t) &= \bar{A}_c x(t) + \bar{B}_c u(t) \\ y(t) &= C_c x(t), \end{aligned} \tag{3.8}$$

where

$$\bar{A}_c = A_c + \Delta A_c, \bar{B}_c = B_c + \Delta B_c, \tag{3.9}$$

with A_c, B_c, C_c, $x(t)$, $y(t)$, $u(t)$ as introduced in the model of Eq. 3.1. $\Delta A_c \in \mathbb{A}_c \subseteq \mathbb{R}^{n\times n}$ and $\Delta B_c \in \mathbb{B}_c \subseteq \mathbb{R}^{n\times 1}$ are continuous-time time-invariant matrices with unknown values representing model uncertainties. $\mathbb{A}_c$ and $\mathbb{B}_c$ represent the sets of possible values of the uncertainties.

3.5.2 Discrete-time representation of continuous-time uncertainties

We consider the following structure for the discrete-time representation of Eq. 3.8:

$$\begin{aligned} x_{k+1} &= (A_d + \Delta A_d)x_k + (B_d + \Delta B_d)u_{k-\gamma} \\ y_k &= C_c x_k, \end{aligned} \tag{3.10}$$

where $A_d \in \mathbb{R}^{n\times n}$, $B_d \in \mathbb{R}^{n\times 1}$, $x_k \in \mathbb{R}^{n\times 1}$, $u_{k-\gamma} \in \mathbb{R}$, and $y_k \in \mathbb{R}$ are defined in Eq. 2.7 while $\Delta A_d \in \mathbb{A}_d \subseteq \mathbb{R}^{n\times n}$ and $\Delta B_d \in \mathbb{B}_d \subseteq \mathbb{R}^{n\times 1}$ are matrices representing discrete-time time-invariant model uncertainties. Recall from Section 2.3.5 that this discretization uses the delayed control input $u_{k-\gamma}$. The uncertainty matrices ΔA_d and ΔB_d capture the continuous-time uncertainties in the discrete-time domain. We define

$$\mathbb{A}_d = \{\Delta A_d(\Delta A_c) \mid \Delta A_c \in \mathbb{A}_c\},$$
$$\mathbb{B}_d = \{\Delta B_d(\Delta A_c, \Delta B_c) \mid \Delta A_c \in \mathbb{A}_c \wedge \Delta B_c \in \mathbb{B}_c\},$$

where ΔA_d and ΔB_d are found using Definitions 3.1 and 3.2 below, for known values of ΔA_c and ΔB_c.

Definition 3.1. ***Computation of*** ΔA_d ***for a known*** ΔA_c***:*** *Consider the systems with uncertainties in Eq. 3.8 and Eq. 3.13.* ΔA_d *for a known* ΔA_c *is computed by:*

$$\Delta A_d(\Delta A_c) = e^{(A_c+\Delta A_c)h} - e^{A_c h}. \tag{3.11}$$

Definition 3.2. ***Computation of*** ΔB_d ***for known*** ΔA_c ***and*** ΔB_c***:*** *Consider the systems with uncertainties in Eq. 3.8 and Eq. 3.13.* ΔB_d *for a known* ΔA_c *and* ΔB_c *is computed by:*

$$\Delta B_d(\Delta A_c, \Delta B_c) = \int_0^h e^{(A_c+\Delta A_c)s}(B_c + \Delta B_c)ds - \int_0^h e^{A_c s} B_c ds. \tag{3.12}$$

3.5.3 Discrete-time model with worst-case uncertainties

To compute the uncertainties of Eq. 3.10 using Definitions 3.1 and 3.2, a single value of the continuous-time uncertainties ΔA_c and ΔB_c is needed. However, the exact value of these matrices is unknown but bounded by the sets $\mathbb{A}_c$ and $\mathbb{B}_c$. Therefore, to capture all possible representations of Eq. 3.10 in the discrete-time domain, the following model is used:

$$\begin{aligned} x_{k+1} &= (A_d + \Delta A_{d,wc})x_k + (B_d + \Delta B_{d,wc})u_{k-\gamma} \\ y_k &= C_c x_k, \end{aligned} \tag{3.13}$$

where $\Delta A_{d,wc} \in \mathbb{A}_d \subseteq \mathbb{R}^{n\times n}$ and $\Delta B_{d,wc} \in \mathbb{B}_d \subseteq \mathbb{R}^{n\times 1}$ are the discrete-time uncertainties with the largest Euclidean norm, i.e., the worst-case uncertainties. These matrices are defined as:

$$\begin{aligned} \Delta A_{d,wc} = & \underset{\Delta A_d \in \mathbb{A}_d}{\arg\max} ||\Delta A_d||, \\ \Delta B_{d,wc} = & \underset{\Delta B_d \in \mathbb{B}_d}{\arg\max} ||\Delta B_d||. \end{aligned} \tag{3.14}$$

The solution of Eq. 3.14 must guarantee that continuous-time uncertainties are captured in the discrete-time domain. However, because of the non-linear nature of the discretization ΔA_d and ΔB_d, finding $\Delta A_{d,wc}$ and $\Delta B_{d,wc}$ that satisfy Eq. 3.14 for any continuous-time uncertainties (i.e., any set $\mathbb{A}_c$, $\mathbb{B}_c$) is a non-trivial task. It might even be the case that $\Delta A_{d,wc}$ and $\Delta B_{d,wc}$ are not uniquely defined. Therefore, an approximation strategy for solving Eq. 3.14 is later explained in Section 3.6, taking into account a specific case of continuous-time uncertainties.

3.5.4 Augmented non-delayed model of pipelined systems with uncertainties

To apply the robustness analysis of Section 3.7, a non-delayed transformation of Eq. 3.13 is needed. To do so, we define augmented state vector using Eq. 2.9, which yields the following augmented system:

$$\begin{aligned} z_{k+1} &= \bar{\Phi}_d z_k + \Gamma_d u_k \\ y_k &= C_d z_k, \end{aligned} \tag{3.15}$$

with

$$\bar{\Phi}_d = \Phi_d + \Delta\Phi_d,$$

where Φ_d, Γ_d and C_d are defined in the model of Eq. 3.4. $\Delta\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ is the discrete-time augmented uncertainty matrix defined in the next subsections. Note that Γ_d is not influenced by the model uncertainties.

3.5.5 Structure of the model uncertainties

To apply the robustness analysis technique of Section 3.7, the model uncertainties of Eq. 3.15 must have the following structure:

$$\Delta\Phi_d = \alpha DGE, \tag{3.16}$$

where $D \in \mathbb{R}^{(n+\gamma)\times n}$ and $E \in \mathbb{R}^{n\times(n+\gamma)}$ are known constant matrices, and $G \in \mathbb{R}^{n\times n}$ is an uncertainty matrix bounded by

$$G^T G \preceq I. \tag{3.17}$$

$\alpha \in \mathbb{R}^+$ is a positive scalar which is used as a scaling factor later in the design. A larger α implies higher robustness.

3.5.6 Uncertainty bound

The uncertainty matrix $\Delta\Phi_d$ described in Section 3.5.5 has an upper bound that can be derived from the assumption that $G^T G \leq I$. Notice that this constraint also implies that the Euclidean norm $||G|| \leq 1$. We derive an upper bound on the uncertainties as

$$||\Delta\Phi_d|| = ||\alpha DGE|| \leq |\alpha|\ ||D||\ ||G||\ ||E|| \leq |\alpha|\ ||D||\ ||E||. \tag{3.18}$$

Eq. 3.18 provides an upper bound on the maximum norm of the uncertainties that are tolerated by the analysis and design method presented in the next sections. This upper bound depends on α, D, and E. α is maximized during the optimization analysis to obtain the maximum upper bound on the uncertainties. D and E are constant matrices that need to be computed before the optimization procedure. Note that an incorrect selection of these matrices (i.e., matrices with smaller norm than the real ones) results in a larger maximum α during the optimization analysis. Therefore, D and E should represent the largest possible norm (the worst-case) of the continuous-time uncertainties.

3.5.7 Selection of D and E

The matrices D and E are to be defined based on the worst-case discrete-time uncertainty. Therefore, the following structure for these matrices is proposed:

$$D = \begin{bmatrix} I \\ 0 \end{bmatrix}, E = \begin{bmatrix} \Delta A_{d,wc} & \Delta B_{d,wc} & 0 \end{bmatrix}. \tag{3.19}$$

Including the definition of D and E in Eq. 3.16, we obtain the uncertainty structure:

$$\Delta\Phi_d = \alpha DGE = \begin{bmatrix} \alpha G\Delta A_{d,wc} & \alpha G\Delta B_{d,wc} & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.20}$$

The method presented in this section provides a strategy to model discrete-time norm-bounded uncertainties in pipelined systems. The resulting discrete-time uncertainties are then used to analyse the system robustness in Section 3.7. In the following section, we present a strategy to approximate the discrete-time uncertainties.

3.6 Model-uncertainty approximation

This section presents an approximation strategy for the uncertainty matrices $\Delta A_{d,wc}$ and $\Delta B_{d,wc}$. In practice, computation of exact $\Delta A_{d,wc}$ and $\Delta B_{d,wc}$ is hard since it requires evaluation of all (infinitely many) possible values in the sets $\mathbb{A}_c$ and $\mathbb{B}_c$. We therefore rely on a numeric method for approximating the discrete-time uncertainties, which is based on a restricted class of continuous-time uncertainties. We consider the case where only one element of ΔA_c and ΔB_c is non-zero with given minimum and maximum values.

Let for any matrix X, $X^{(i,j)}$ represent the matrix element located in the i^{th} row and j^{th} column. Let $\Delta A_c(i,j,a)$ be a matrix with element (i,j) equal to a, for some $a \in \mathbb{R}$, and all other elements equal to 0. That is, $\Delta A_c(i,j,a)^{(i,j)} = a$ and for all $k,l \in \mathbb{N}$ with $1 \le k,l \le n$ such that $(k,l) \neq (i,j)$, $\Delta A_c(i,j,a)^{(k,l)} = 0$. This allows us to define the set $\mathbb{A}_c$ introduced in Section 3.5.1 as follows, for given indexes $i,j \in \mathbb{N}$ with $1 \le i,j \le n$ and a maximum uncertainty $a_{max} \in \mathbb{R}^{\geq}$:

$$\mathbb{A}_c = \left\{\Delta A_c(i,j,a) \,\middle|\, a \in [-a_{max}, a_{max}]\right\}.$$

Similarly, let $\Delta B_c(o,b)$ be a matrix with element $(o,1)$ equal to b, for some $b \in \mathbb{R}$, and all other elements equal to 0. That is, $\Delta B_c(o,b)^{(o,1)} = b$ and for all $p \in \mathbb{N}$ with $1 \le p \le n$ such that $p \neq o$, $\Delta B_c(o,b)^{(p,1)} = 0$. The set $\mathbb{B}_c$ is defined as follows, for a given index $o \in \mathbb{N}$ with $1 \le o \le n$ and a maximum uncertainty $b_{max} \in \mathbb{R}^{\geq}$:

$$\mathbb{B}_c = \left\{\Delta B_c(o,b) \mid b \in [-b_{max}, b_{max}]\right\}.$$

This particular structure of the model uncertainties is used because numerically we have observed that it is possible to estimate the worst-case discrete-time uncertainties by evaluating the continuous-time uncertainties for the boundaries $-a_{max}$, a_{max}, $-b_{max}$, and b_{max}. To make this precise, we define the auxiliary sets $\hat{\mathbb{A}}_d$ and $\hat{\mathbb{B}}_d$ as:

$$\begin{aligned}\hat{\mathbb{A}}_d &= \left\{\Delta A_d\left(\Delta A_c(i,j,a)\right) \,\middle|\, a \in \{-a_{max}, a_{max}\}\right\},\\ \hat{\mathbb{B}}_d &= \left\{\Delta B_d\left(\Delta A_c(i,j,a), \Delta B_c(o,b)\right) \,\middle|\, a \in \{-a_{max}, a_{max}\} \wedge b \in \{-b_{max}, b_{max}\}\right\}.\end{aligned}$$

Then, Eq. 3.14 is approximated as

$$\begin{aligned}\Delta A_{d,wc} \approx\ & \underset{\Delta A_d \in \hat{\mathbb{A}}_d}{\operatorname{argmax}} ||\Delta A_d||,\\ \Delta B_{d,wc} \approx\ & \underset{\Delta B_d \in \hat{\mathbb{B}}_d}{\operatorname{argmax}} ||\Delta B_d||.\end{aligned} \tag{3.21}$$

Numerically, we have also observed that upscaling the discrete-time uncertainties by a scalar corresponds to upscaled continuous-time uncertainties, if the particular structure of model uncertainties is used. That is, consider the discrete-time uncertainties $\Delta A_{d,1}(\Delta A_{c,1})$ and $\Delta A_{d,2}(\Delta A_{c,2})$ derived from some continuous-time uncertainties $\Delta A_{c,1}$, $\Delta A_{c,2} \in \mathbb{A}_c$. If $\Delta A_{d,2} = \alpha \Delta A_{d,1}$ and $\alpha \ge 1$, then $||\Delta A_{c,2}|| \ge ||\Delta A_{c,1}||$.

This upscaling correspondence is of importance because the robustness-analysis technique finds the largest α that guarantees a stable system. This implies that a controller that can tolerate an $\alpha \geq 1$ in the discrete-time domain, can tolerate the original set of continuous-time uncertainties (or greater). Note that to verify this upscaling correspondence, an approximation of the discrete-time uncertainties in the continuous-time domain is required. An approximation of ΔA_c can be found from $A_d + \Delta A_d = e^{A_c + \Delta A_c}$. This requires (among others) the computation of a natural logarithm of a matrix. The logarithm can be approximated using a Mercator-series expansion [62, Chapter 11]. Likewise to find ΔB_c, we apply a Taylor-series expansion to the integral term of $B_d + \Delta B_d = \int_0^h e^{\bar{A}_c s} \bar{B}_c ds$ [104, Chapter 1.2]. ΔB_c can then be found from the right-hand side of the resulting expansion. It would be interesting to investigate an analytical argument that would support the observed scaling between the discrete-time and continuous-time uncertainties.

Example 3.1. ***Uncertainty approximation:*** *Using Example 2.1, we define the uncertainties as:*

$$\Delta A_c = \begin{bmatrix} 0 & 0 \\ 0 & \Delta A_c^{(2,2)} \end{bmatrix}, \Delta B_c = \begin{bmatrix} 0 \\ \Delta B_c^{(2,1)} \end{bmatrix},$$

where $a_{max} = 0.05$ *and* $b_{max} = 0.2$, *and thus*

$$-0.05 \leq \Delta A_c^{(2,2)} \leq 0.05$$
$$-0.2 \leq \Delta B_c^{(2,1)} \leq 0.2.$$

These assumptions correspond to an uncertainty of 2% *in one element in the nominal matrices. Definitions 3.1 and 3.2 are used to find one* ΔA_d *and one* ΔB_d *for each possible continuous-time uncertainties* ΔA_c *and* ΔB_c. *The continuous-time uncertainties that produce the discrete-time matrices with the largest norm are:*

$$\Delta A_c = \begin{bmatrix} 0 & 0 \\ 0 & 50 \end{bmatrix} \times 10^{-3}, \Delta B_c = \begin{bmatrix} 0 \\ 200 \end{bmatrix} \times 10^{-3}. \tag{3.22}$$

The uncertainties with the worst-case norm are then:

$$\Delta A_{d,wc} \approx \begin{bmatrix} -9.3 & 72.7 \\ -370.8 & 1879 \end{bmatrix} \times 10^{-6}, \Delta B_{d,wc} \approx \begin{bmatrix} 312.3 \\ 8379.3 \end{bmatrix} \times 10^{-6}.$$

For the case of two pipes ($\gamma = 2$), *we then use Eq. 3.19 to derive D and E:*

$$E = \begin{bmatrix} -9.3 & 72.7 & 312.3 & 0 \\ -370.8 & 1879 & 8379.3 & 0 \end{bmatrix} \times 10^{-6}, D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

The method presented in this section provides a strategy to model pipelined systems with model uncertainties. In the next section, we use such a model to analyse the robustness of an existing performance-oriented controller or to design a robustness-oriented controller.

3.7 Robustness of pipelined systems

We use a control law of the form presented in Eq. 3.5. The gains K and F need to be analysed or redesigned for the model with uncertainties described in Eq. 3.15 for guaranteeing closed-loop stability. Note that F does not influence stability but it is important for performance benchmarking experiments, which are performed for a given robust feedback gain K. In the following, we focus on analysing and designing a feedback gain K with respect to the model uncertainties described above to guarantee closed-loop stability.

Using $u_k = Kz_k$, the closed-loop model for the system of Eq. 3.15 is the following:

$$z_{k+1} = \bar{\Phi}_{cl} z_k, \tag{3.23}$$

with $\bar{\Phi}_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ given by

$$\bar{\Phi}_{cl} = \Phi_{cl} + \Delta\Phi_d, \tag{3.24}$$

where $\Phi_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ is the nominal closed-loop system further given by

$$\Phi_{cl} = \Phi_d + \Gamma_d K. \tag{3.25}$$

In the following, we focus on the following questions:

- **Robustness analysis for a performance-oriented controller:** given a feedback gain K that stabilizes the nominal closed-loop system in Eq. 3.25, find the maximum α for which closed-loop stability of the system with uncertainties in Eq. 3.23 can be guaranteed.
- **Robustness-oriented controller design:** design a feedback gain K while maximizing α such that closed-loop stability of the system with uncertainties in Eq. 3.23 can be guaranteed.

To analyse robustness or redesign a pipelined control system against norm-bounded time-invariant uncertainties, Lemmas 3.1 to 3.3 below are used. Then, Theorem 3.1 presents a robustness-analysis technique for an already designed controller and Theorem 3.2 presents a control-design technique that maximizes tolerable uncertainties. These theorems are adapted from [137, 47] to pipelined systems.

3.7.1 Preliminary lemmas

Lemma 3.1. *[139] Given constant matrices Θ, Λ, and Ξ of appropriate dimensions with $\Theta = \Theta^T$, and a matrix G such that $G^T G \preceq I$, inequality*

$$\Theta + \Lambda G \Xi + (\Lambda G \Xi)^T \prec 0$$

holds if and only if there exists a scalar $\epsilon \in \mathbb{R}^+$ such that

$$\Theta + \epsilon^{-1} \Lambda \Lambda^T + \epsilon \Xi^T \Xi \prec 0.$$

Lemma 3.2. ***Schur complement [147].*** *Given matrices M, L, and P, with $M = M^T$, and a positive-definite matrix $P = P^T \succ 0$, the matrix inequality $M + L^T P^{-1} L \prec 0$ can be rewritten in the following form:*

$$\begin{bmatrix} M & L^T \\ L & -P \end{bmatrix} \prec 0. \tag{3.26}$$

Lemma 3.3. *[44, 28] The system in Eq. 3.23 is globally asymptotically stable if there exist a matrix $O \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ such that $O = O^T \succ 0$ (i.e., positive definite), and a matrix $V \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ such that*

$$\begin{bmatrix} O & \bar{\Phi}_{cl} V \\ (\bar{\Phi}_{cl} V)^T & V^T + V - O \end{bmatrix} \succ 0. \tag{3.27}$$

3.7.2 Robustness analysis for performance-oriented controller

The following theorem finds the maximum scalar α that stabilizes the system of Eq. 3.23 with a given feedback gain K. α is therefore used to quantify the controller robustness for a particular resource configuration. A larger α implies a higher system robustness.

Theorem 3.1. ***Robustness analysis (adapted from [137, 47]):*** *Consider the closed-loop system with uncertainties of Eq. 3.23, where the nominal closed-loop matrix $\Phi_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ is given by Eq. 3.25, and the uncertainty matrix $\Delta\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ is given by Eq. 3.16 with known matrices $D \in \mathbb{R}^{(n+\gamma)\times n}$ and $E \in \mathbb{R}^{n\times(n+\gamma)}$, some to-be-determined $\alpha \in \mathbb{R}^+$, and some uncertainty matrix $G \in \mathbb{R}^{n\times n}$, such that $G^T G \preceq I$. The system in Eq. 3.23 is globally asymptotically stable for a previously designed $K \in \mathbb{R}^{1\times(n+\gamma)}$, if there exist a positive-definite matrix $O \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$, a matrix $V \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$, and* *scalars $\lambda \in \mathbb{R}^+$ and $\sigma \in \mathbb{R}^+$ that satisfy*

$$O \succ 0, \tag{3.28}$$

$$\begin{bmatrix} -O & \lambda D & -\Phi_{cl}V & 0 \\ \lambda D^T & -\sigma\lambda I & 0 & 0 \\ -(\Phi_{cl}V)^T & 0 & O-V-V^T & (EV)^T \\ 0 & 0 & (EV) & -\lambda I \end{bmatrix} \prec 0. \tag{3.29}$$

The uncertainty matrix $\Delta\Phi_d$ is then given by αDE with $\alpha = \sigma^{-0.5}$. Moreover, the maximum uncertainties (i.e., the maximum α) that the system of Eq. 3.23 can tolerate is found by solving the following optimization problem:

minimize *σ*
subject to *Eq. 3.28 and Eq. 3.29.*

Proof. For stability of the system defined in Eq. 3.23, we start with using Lemma 3.3. There exist a positive-definite matrix $O \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ and a matrix $V \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ such that

$$\begin{bmatrix} O & \bar{\Phi}_{cl}V \\ (\bar{\Phi}_{cl}V)^T & V^T+V-O \end{bmatrix} \succ 0.$$

Substituting $\bar{\Phi}_{cl}$ in Eq. 3.24, using Eq. 3.16 and multiplying by -1 yields:

$$\begin{bmatrix} -O & -\Phi_{cl}V-\alpha DGEV \\ -(\Phi_{cl}V)^T-\alpha(DGEV)^T & O-V-V^T \end{bmatrix} \prec 0.$$

This is equivalent to:

$$\begin{bmatrix} -O & -\Phi_{cl}V \\ -(\Phi_{cl}V)^T & O-V-V^T \end{bmatrix} + \begin{bmatrix} 0 & -\alpha DGEV \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\alpha(DGEV)^T & 0 \end{bmatrix} \prec 0.$$

Double transposing the third matrix above yields:

$$\begin{bmatrix} -O & -\Phi_{cl}V \\ -(\Phi_{cl}V)^T & O-V-V^T \end{bmatrix} + \begin{bmatrix} 0 & -\alpha DGEV \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -\alpha DGEV \\ 0 & 0 \end{bmatrix}^T \prec 0.$$

Decomposing the second and third matrices gives:

$$\begin{bmatrix} -O & -\Phi_{cl}V \\ -(\Phi_{cl}V)^T & O-V-V^T \end{bmatrix} + \begin{bmatrix} -\alpha D \\ 0 \end{bmatrix} G \begin{bmatrix} 0 & EV \end{bmatrix} + \left(\begin{bmatrix} -\alpha D \\ 0 \end{bmatrix} G \begin{bmatrix} 0 & EV \end{bmatrix} \right)^T \prec 0.$$

Applying Lemma 3.1, we obtain:

$$\begin{bmatrix} -O & -\Phi_{cl}V \\ -(\Phi_{cl}V)^T & O-V-V^T \end{bmatrix} + \epsilon^{-1}\begin{bmatrix} -\alpha D \\ 0 \end{bmatrix}\begin{bmatrix} -\alpha D^T & 0 \end{bmatrix} + \epsilon\begin{bmatrix} 0 \\ (EV)^T \end{bmatrix}\begin{bmatrix} 0 & EV \end{bmatrix} \prec 0.$$

Note that in the above condition, the second and third element capture the influence of the uncertainties on the overall system stability. These terms depend on the free matrices D, E, and V. Therefore, it is not possible to generalize their negative or positive definiteness. The first term captures the influence on the stability of the nominal closed-loop system. Since the closed-loop nominal system is stable, this term must be negative definite as shown in Lemma 3.3. The last expression no longer depends on G.

Further,

$$\begin{bmatrix} -O & -\Phi_{cl}V \\ -(\Phi_{cl}V)^T & O-V-V^T \end{bmatrix} + \begin{bmatrix} \epsilon^{-1}\alpha^2 DD^T & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \epsilon(EV)^T EV \end{bmatrix} \prec 0,$$

which is equivalent to

$$\begin{bmatrix} -O+\epsilon^{-1}\alpha^2 DD^T & -\Phi_{cl}V \\ -(\Phi_{cl}V)^T & O-V-V^T+\epsilon(EV)^T EV \end{bmatrix} \prec 0. \tag{3.30}$$

Applying the Schur complement (Lemma 3.2) with respect to the term $-O+\epsilon^{-1}\alpha^2 DD^T$ gives:

$$\begin{bmatrix} -O & \epsilon^{-1}D & -\Phi_{cl}V \\ \epsilon^{-1}D^T & -\epsilon^{-1}\alpha^{-2}I & 0 \\ -(\Phi_{cl}V)^T & 0 & O-V-V^T+\epsilon(EV)^T EV \end{bmatrix} \prec 0.$$

Applying the Schur complement on the term $O-V-V^T+\epsilon(EV)^T EV$ then gives:

$$\begin{bmatrix} -O & \epsilon^{-1}D & -\Phi_{cl}V & 0 \\ \epsilon^{-1}D^T & -\epsilon^{-1}\alpha^{-2}I & 0 & 0 \\ -(\Phi_{cl}V)^T & 0 & O-V-V^T & (EV)^T \\ 0 & 0 & EV & -\epsilon^{-1}I \end{bmatrix} \prec 0.$$

Defining $\lambda = \epsilon^{-1}$ and $\sigma = \alpha^{-2}$, the above is equivalent to Eq. 3.29 completing the proof. By finding the minimum σ satisfying the above Linear Matrix Inequality (LMI), we obtain the maximum α (scaling factor of robustness). $\square$

Note that in Theorem 3.1 (and also in Theorem 3.2 later), minimizing σ maximizes α, which means that the uncertainties that can be handled are also maximized.

Remark 3.1. *The conditions shown in Theorem 3.1 (and also in Theorem 3.2 later) correspond to a non-linear matrix inequality due to the multiplication of the terms σ and λ. However, assuming a value of $\lambda > 0$, the conditions are simplified to a Linear Matrix Inequality. Therefore, we assign different values to λ between a small number (e.g., 100×10^{-27}) and a large number (e.g., 10) that produce a feasible solution. We solve each of the resulting optimization problems and we then report the largest α. This strategy does not guarantee that the found α is optimal due to the discretization of the design space of λ.*

Remark 3.2. *Applying the Schur complement (Lemma 3.2) to one element of a block matrix (e.g., Eq. 3.30) divides one element into four new elements. As a result, a new row and a new column are created inside the matrix inequality, which are filled by the newly created elements and zeros. The organization of such elements might differ as long as they are placed according to the symmetry of the matrix inequality. It is possible to prove that two matrix inequalities L_1 and L_2 with elements placed symmetrically in different positions are equivalent by finding a transformation matrix F such that $F^T L_1 F = L_2$.*

Remark 3.3. *Theorem 3.1 is applicable to time-invariant (e.g., [149, 105]) and time-variant systems (e.g., [137, 47]). However, the method for approximating continuous-time uncertainties presented in Section 3.6 is only applicable to time-invariant systems. Therefore, the overall strategy presented in this chapter remains only applicable to time-invariant systems. An alternative approximation method suited for time-variant systems remains an interesting research question.*

Example 3.2. ***Robustness analysis of a performance-oriented controller:*** *Consider the model from Example 2.2, the controller from Example 2.5, and the uncertainties from Example 3.1:*

$$E = \begin{bmatrix} -9.3 & 72.7 & 312.3 & 0 \\ -370.8 & 1879 & 8379.3 & 0 \end{bmatrix} \times 10^{-6}, D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Consider that the minimum α required (i.e., a robustness constraint) for this example is an $\alpha \geq 1$. This implies that the controller can tolerate uncertainties at least larger than the product of E and D. The optimization problem of Theorem 3.1 is then formulated. Using Remark 3.1, the optimization problem is converted into a set of Linear Matrix Inequalities. The modelling tool Yalmip [83] together with the convex optimization software SDPT3 [118] are used to solve the optimization problem instances. As a *result, $\alpha = 7.7$ is found for $\gamma = 2$. Given that $\alpha \geq 1$, the robustness constraint of our problem is met for two pipes.*

3.7.3 Robustness-oriented pipelined controller design

The following theorem finds the feedback gain K that maximizes the scalar α, while guaranteeing the stability of the system of Eq. 3.23. Like in the previous case, a larger α implies a more robust controller.

Theorem 3.2. ***Robust controller design (adapted from [137, 47]):*** *Consider the closed-loop system with uncertainties of Eq. 3.23, where the nominal closed-loop matrix $\Phi_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ is given by Eq. 3.25, and the uncertainty matrix $\Delta\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$ is given by Eq. 3.16 with known matrices $D \in \mathbb{R}^{(n+\gamma)\times n}$ and $E \in \mathbb{R}^{n\times(n+\gamma)}$, some to-be-determined $\alpha \in \mathbb{R}^+$, and some uncertainty matrix $G \in \mathbb{R}^{n\times n}$, such that $G^T G \preceq I$. The system in Eq. 3.23 is globally asymptotically stable, if there exist a positive-definite matrix $O \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$, matrices $K \in \mathbb{R}^{1\times(n+\gamma)}$, $V \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$, and $V_K \in \mathbb{R}^{1\times(n+\gamma)}$, and scalars $\lambda \in \mathbb{R}^+$ and $\sigma \in \mathbb{R}^+$ that satisfy*

$$O \succ 0, \tag{3.31}$$

$$\begin{bmatrix} -O & \lambda D & -\Phi_d V - \Gamma_d V_K & 0 \\ \lambda D^T & -\sigma\lambda I & 0 & 0 \\ -(\Phi_d V + \Gamma_d V_K)^T & 0 & O - V - V^T & (EV)^T \\ 0 & 0 & (EV) & -\lambda I \end{bmatrix} \prec 0, \tag{3.32}$$

with $V_K = KV$ and $K = V_K V^{-1}$. The uncertainty matrix $\Delta\Phi_d$ is then given by αDE with $\alpha = \sigma^{-0.5}$. Moreover, the uncertainties that the system of Eq. 3.23 can tolerate is maximized by solving the following optimization problem:

$$\textbf{minimize } \sigma$$
$$\textbf{subject to } \text{Eq. 3.31 and Eq. 3.32.}$$

Proof. The proof follows the steps of the proof of Theorem 3.1 using the definition of Φ_{cl} (Eq. 3.25) and $V_K = KV$. The details are therefore omitted. The invertibility of V is implied because from the matrix inequality it can be deduced that $V + V^T \succ O \succ 0$, which means that V is full rank and therefore invertible. □

3.8 Robustness example of pipelined systems

The robustness-analysis procedure of Section 3.7 quantifies the robustness of a pipelined controller by means of the scalar α. However, in pipelined control, each resource configuration has different uncertainties in the discrete-time model . Therefore, the robustness analysis of a performance-oriented controller or the design of a robustness-oriented controller has to be repeated for each resource configuration to be considered. This procedure allows for the performance-oriented design to select a resource configuration that improves QoC while meeting a robustness constraint (i.e., minimum α). For the robustness-oriented control design, it allows to select a resource configuration that provides a controller with maximized robustness. This is best explained by means of the following examples.

Example 3.3. ***Robustness analysis of performance-oriented controllers:*** *Continuing with Example 3.2, the robustness analysis is applied to a range of resource configurations from one to eight pipes i.e.,* $\gamma = 1, \ldots, 8$*. Fig. 3.2 shows the maximum* α *found (using Theorem 3.1) in each one of the resource configurations. Taking into account that the robustness requirement is* $\alpha \geq 1$*, we observe the following from Fig. 3.2:*

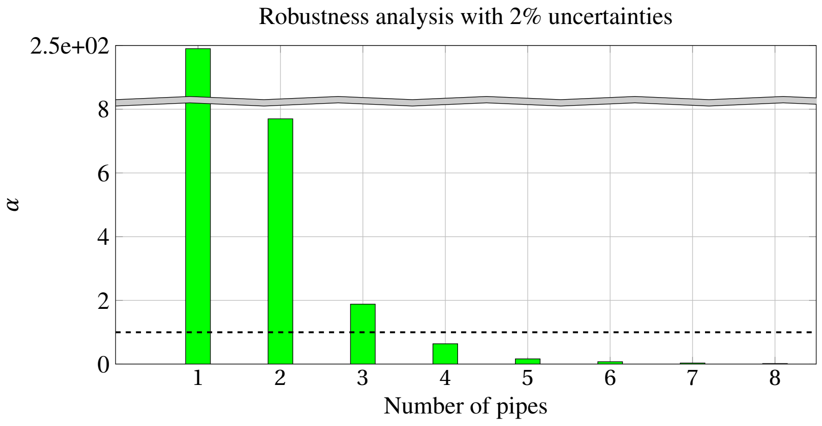


Figure 3.2: Robustness analysis of motivational example.

- *From Example 2.5, it was concluded that the resource configuration that gives a meaningful improvement in QoC is four pipes. However, the robustness constraint of* $\alpha \geq 1$ *is only met in the resource configurations* $\gamma \in \{1, \ldots, 3\}$*. Therefore, the resource configuration that gives a meaningful improvement in settling time while guaranteeing the robustness constraint is three pipes.*
- *Each newly added pipe decreases the maximum* α *that the controller can tolerate, because increasing the number of pipes produces a more aggressive controller response, which is more likely to become unstable with model uncertainties.*

Table 3.1: Settling times (ms) of motivational example with different values of uncertainties

Number of pipes	$\Delta A_c^{(2,2)} = 0$ $\Delta B_c^{(2,1)} = 0$	$\Delta A_c^{(2,2)} = -0.0025$, $\Delta B_c^{(2,1)} = 0.010$	$\Delta A_c^{(2,2)} = 0.0025$ $\Delta B_c^{(2,1)} = -0.010$
1	222.2	247.5	313.5
2	153.2	165.8	198.7
3	130.5	138.0	161.8
4	119.3	robustness constraint not met, i.e., $\alpha < 1$	
5	111.9		
6	107.5		
7	104.1		
8	101.8		

- *Already a* 2% *uncertainty in one of the elements of the model matrices has a major impact on the overall system robustness. This is because the B_c matrix has only one non-zero element. An uncertainty in such an element directly affects the amount of energy that the controller inputs to the dynamic system, affecting the stability of the system.*

- *The controller design of Section 3.3.2 optimizes QoC for a nominal system. However, the uncertainties in the system may deteriorate the QoC. Since the exact value of the uncertainties is not known but only a range is known, the settling time varies for different values of uncertainties. Table 3.1 shows an example of such performance deterioration. The settling time increases with the model uncertainties. However, the best performance is still achieved with the highest number of pipes. The settling times of the systems with uncertainties and resource configurations* $4\ldots8$ *are not shown because* $\alpha < 1$ *(meaning the robustness constraint is not met).*

Example 3.4. ***Robustness-oriented pipelined control design:*** *A controller that maximizes α is designed using Theorem 3.2. Fig. 3.3 shows the resulting α when the initial uncertainties assumed in Example 3.1 was increased to* 50%. *We observe the following from the graphs:*

- *The maximum α that the system can tolerate increases with the number of pipes, which potentially means that the robustness of the pipelined controller increases with each newly added pipe when designed for maximum robustness. This is possible due to the higher sampling rate.*

- *The new values of α are significantly larger than the values found with Theorem 3.1, which means a more robust controller is obtained. However, the increased robustness comes at the cost of performance deterioration. For example, in the nominal system with one sensing pipe, the settling time of a controller designed for robustness (using Theorem 3.2) is* 2 *s, while using the PSO algorithm and not considering the uncertainties in the controller design gives* $222\ ms$*.*
- *The controller resulting from Theorem 3.2 is mostly interesting for systems with large uncertainties, where performance-oriented designs may not be feasible. For example, in ADAS, uncertainties may originate from changing traffic conditions that are difficult to predict. There, a more robust controller is desired since it increases vehicle safety. The most robust controller corresponds to the hardware configuration with the largest number of pipes.*

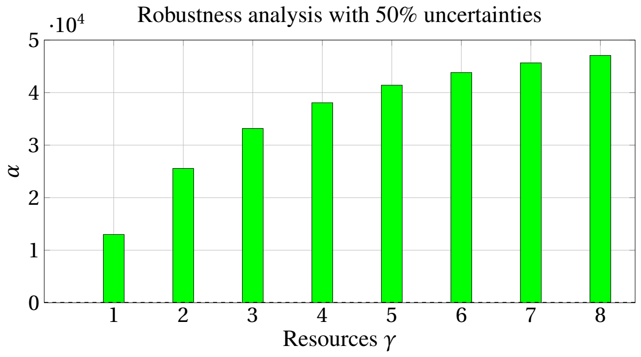


Figure 3.3: Robustness analysis including the controller design.

3.9 Feasibility study

3.9.1 Overview

In this section, we demonstrate the effectiveness of a performance-oriented pipelined controller using Hardware-In-the-Loop (HIL) simulation. This simulation consists of emulating the plant behaviour implementing the controller and the model in independent embedded hardware, such that there is an exchange of electrical signals between the plant and the controller (see [65, 12],[84, Chapter 14] and the references therein).

HIL offers the possibility of studying the behaviour of the pipelined controller in a realistic scenario, e.g., when multiple processing resources exchange their previous controller outputs to compute new controller outputs, or when changing the nominal model of the system to evaluate the impact of model uncertainties in the controller.

We subdivide the remainder of this section in four parts. We introduce an application where pipelined systems can be of benefit. We then present our HIL simulation. Next, we apply the design flow of Section 3.4 considering the available hardware and the presented example. We finally present some results of our HIL simulation.

3.9.2 Plant description and model

We consider an example of an assembly line like the xCPS system shown in Fig. 2.1 [2]. This kind of system mimics the behaviour of complex machines such as assembly lines or wafer scanners, in which data-intensive algorithms (e.g., image processing) are used as sensors in control loops.

In xCPS, the speed of assembly objects travelling on the belt has to be regulated such that the objects move slowly when they have to be processed by one of the actuators in front of the belts, and fast when they have to travel between actuators. The conveyor belt rotation speed ω is controlled by a continuous-time PI controller. The settling time of such a controller is 20 ms. The PI controller does not receive information about the object speed moving on top of the conveyor belt, but only about the belt itself. Ideally, both speeds (i.e., belt and object) should be the same; however, in practise, due to model uncertainties (e.g., unmodelled friction between the block and the belt borders) they might differ. Therefore, a camera and an image-processing algorithm are used as a sensor of an additional image-based controller to regulate the block's speed. The Hough transform for circles is used to recognize the position of the block on the belt, which is used to estimate the block speed. A schematic of the system is shown in Fig. 3.4. The plant to be controlled by the image-based control is the PI controller together with the model of the conveyor belt (i.e., the inner loop). The output of the image-based controller is then used as reference for the PI controller.

Using the worst-case execution time of the image-processing algorithm, the control-computation operation, and the actuation operation, a $\tau = 90$ ms latency is introduced to the image-based control loop, which compared to the 20 ms settling time of the inner loop, potentially limits the performance of the image-based control loop. A pipelined performance-oriented design may improve the performance of the image-based controller and the system performance.

The conveyor belt is moved by a DC motor. The combined model of such a motor

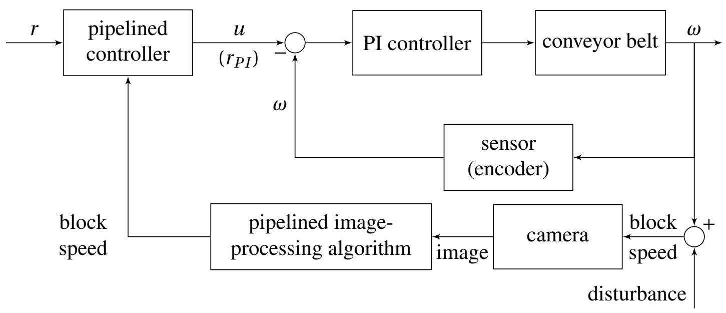


Figure 3.4: Controller structure of one of the belts in the feasibility study.

Table 3.2: Conveyor belt motor parameters.

Symbol	description	value	units
K_m	motor electrical constant	1.25	$^{vs}/_{rad}$
J_m	mechanical inertia	17.44	gm^2
R_m	motor resistance	10.39	Ω
b_m	viscous friction coefficient	0.58	Nms
K_p	controller proportional gain	4.10	
K_I	motor integral gain	166	

and the PI controller (i.e., the inner loop) is given by

$$\begin{bmatrix} \dot{x}^1 \\ \dot{x}^2 \end{bmatrix} = \begin{bmatrix} -K_m^2 - \frac{b_m}{J_m} - \frac{K_p K_m}{R_m J_m} & \frac{K_m K_i}{R_m J_m} \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x^1 \\ x^2 \end{bmatrix} + \begin{bmatrix} \frac{K_p K_m}{R_m J_m} \\ 1 \end{bmatrix} r_{PI},$$

with $\begin{bmatrix} x^1 & x^2 \end{bmatrix}^T = \begin{bmatrix} \omega & \int (r_{PI} - \omega) dt \end{bmatrix}^T$, ω the belt angular speed, and r_{PI} the reference sent to the PI controller. The model parameters are shown in Table 3.2. The resulting matrices are

$$A_c = \begin{bmatrix} -70.36 & 1.14 \times 10^3 \\ -1 & 0 \end{bmatrix}, B_c = \begin{bmatrix} 28.31 \\ 1 \end{bmatrix}.$$

The b_m presented in Table 3.2 corresponds to a nominal value. However, the actual value is estimated in a range of $\Delta b_m = \pm 0.04$ around the nominal b_m. This is caused because the motor is moving a belt whose friction is unknown but bounded to a range.

The bounds on b_m are used to create the uncertainty matrices

$$\Delta A_c = \begin{bmatrix} \Delta A_c^{(1,1)} & 0 \\ 0 & 0 \end{bmatrix}, \Delta B_c = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

where $a_{max} = \frac{\Delta b_m}{J_m}$ and $-2.30 \leq \Delta A_c^{(1,1)} \leq 2.30$.

3.9.3 HIL simulation

3.9.3.1 Platform requirements

In order to make a HIL simulation for pipelined control, we need a platform with multi-core capabilities and a global notion of time. Multiple cores are required to independently run the controller in a pipelined fashion, as well as the plant model. A global notion of time is required to trigger the sensing and actuation operations in each core at precise time instants, so that it guarantees the sampling period, sensing-to-actuating delay and the pipelined parallelism.

Fig. 3.5 shows an example of running a three-core pipelined controller in the plant of Section 3.9.2. The HIL simulation needs four cores in total. The global notion of time allows to trigger a new sensing operation every 30 ms. The total sensing-to-actuating delay corresponds to τ (90 ms in this case) for all the cores. Note that the image-processing algorithm delay may be shorter than its worst-case execution time. Therefore, the global notion of time is used to trigger the actuation operations as close as possible to τ seconds after the start of the sensing operation, achieving pipelined parallelism.

3.9.3.2 CompSOC platform

We use CompSOC, a multi-core tile-based architecture [48] as our implementation platform. In each core, the hardware resources are allocated according to a Time Division Multiplexing (TDM) table. The TDM tables are synchronized using a global clock, i.e., there is a global notion of time.

3.9.3.3 HIL set-up

Our HIL simulation uses the CompSOC platform with one core emulating the inner control loop of Fig. 3.4 and one to three cores running the pipelined controller.

The core emulating the inner loop runs a discrete-time version of the model discretized at a sampling rate $h_p = 500\ \mu s$. h_p satisfies $h_p << h$, implying the plant appears to be continuous from the controller perspective.

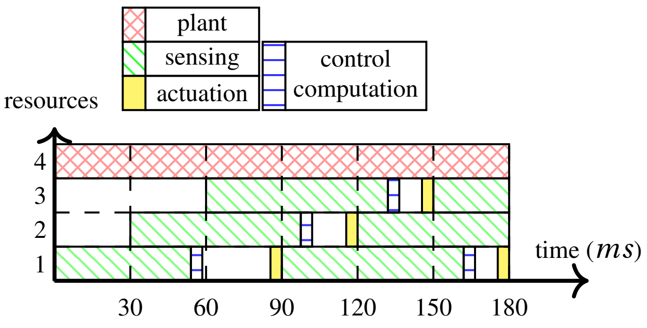


Figure 3.5: Required platform for a three-resources pipelined controller for the plant presented in Section 3.9.2.

In the cores emulating the pipelined controller, the sensing, control computation, and actuation operations are assigned to slots in the TDM tables. The operation allocation has to guarantee that the time elapsed between the start of sensing and the end of actuation slots corresponds to τ. The actuation slot sends the controller input to the plant at the end of the actuation slot to guarantee τ. The operation allocation across TDM tables also needs to guarantee pipelined parallelism, i.e., the start of the sensing operation should be allocated such that pipelined parallelism is achieved. The global clock is then used to align the TDM tables among the cores.

The image-processing algorithm is not implemented in the HIL because of the lack of actual images. The delay of the image-processing algorithm is simulated by increasing the number of slots dedicated to the sensing operation. The sensing information is read in the first sensing slot and delivered to the controller after the last sensing slot.

Table 3.3 shows an example of the TDM tables of a HIL simulation running three sensing pipes for the system of Section 3.9.2. The table length corresponds to τ guaranteeing the sensing-to-actuating delay. The TDM positions are synchronized among the three tables using the global clock. The operations are allocated in such a way that a new sensing operation starts every 30 ms, guaranteeing pipelined parallelism.

3.9.4 Application of the design flow

To find the resource configuration that improves the QoC while guaranteeing a robustness constraint, following Example 3.2, we set the robustness constraint to $\alpha \geq 1$. We apply the design flow of Section 3.4 for the available cores $\gamma = 1 \ldots 3$. We show the analysis for the case of $\gamma = 3$.

Table 3.3: TDM slots for a three-core implementation on the CompSOC platform. S_j, C_j, and A_j with $j = 1,\dots,3$ correspond to the sensing, control computation, and actuation operations in core j, respectively.

Table position	TDM position								
	1	2	3	4	5	6	7	8	9
core 1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	C_1	A_1
core 2	S_2	C_2	A_2	S_2	S_2	S_2	S_2	S_2	S_2
core 3	S_3	S_3	S_3	S_3	C_3	A_3	S_3	S_3	S_3

$\longleftarrow$ 90 ms $\longrightarrow$

$\longleftarrow$ 30 ms $\longrightarrow$

Model discretization: Using the procedure explained in Section 3.3.1, we find a sampling period h and a discrete-time augmented model Φ_d, Γ_d for each resource configuration γ. For the resource configuration $\gamma = 3$, $h = 30$ ms and the model is given by

$$\Phi_d = \begin{bmatrix} -0.0099 & 12.1327 & 0.5648 & 0 & 0 \\ -0.0106 & 0.7349 & 0.0203 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \Gamma_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Performance-oriented controller design: Using the procedure explained in Section 3.3.2, we compute a controller K, F for each resource configuration γ. For $\gamma = 3$, the controller is given by

$$K = \begin{bmatrix} 11.24 & -2.47 & -102.26 & -188.94 & -432.09 \end{bmatrix} \times 10^{-3},$$
$$F = 1.803.$$

The controllers are implemented in Simulink using the nominal plants. For each controller, we measure the settling time produced by a change in the controller reference, which determines the control performance. The resulting trade-off between control performance and resource usage is shown in Fig. 3.6. In this case, each added core significantly improves the settling time. Therefore the optimal resource configuration (without considering the system robustness) is $\gamma = 3$.

Model-uncertainty approximation: The procedure explained in Section 3.6 is applied to the resource configurations. A pair of uncertainty matrices E, D is found for

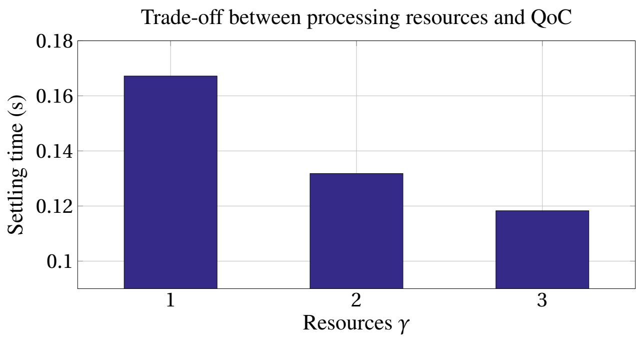


Figure 3.6: Comparison of LQR tuned with different strategies on our motivational example.

each resource configuration. The resulting matrices for $\gamma = 3$ are

$$E = \begin{bmatrix} 0.0037 & 0.2779 & 0 & 0 & 0 \\ -0.0002 & -0.0042 & 0 & 0 & 0 \end{bmatrix}, D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Robustness analysis for performance-oriented control: Applying Theorem 3.1 for the three considered resource configurations, an α for each resource configuration is found. For the case of $\gamma = 3$, $\alpha = 2.77$.

Trade-off analysis: Using the α of each resource configuration, Fig. 3.7 is drawn. Given that all the resource configurations have $\alpha \geq 1$, $\gamma = 3$ is chosen as the resource configuration that improves QoC while guaranteeing a robustness constraint.

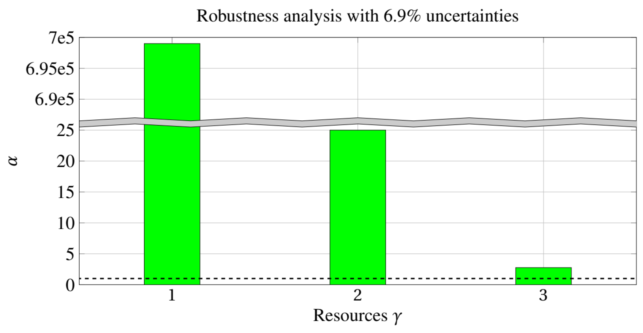


Figure 3.7: Robustness analysis in the feasibility study.

3.9.5 HIL simulation results

We first validate our HIL simulation by comparing it with a Simulink-based simulation. Fig. 3.8 compares controller responses with three pipes. Both simulations show the same result implying that our HIL is correct and that pipelined control is feasible. Fig. 3.8 only shows information every h seconds for the HIL simulation while for the Simulink simulation it shows a continuous line. This is as expected in a realistic scenario, because sensing information is only available when a core finishes the image-processing algorithm, i.e., every h seconds.

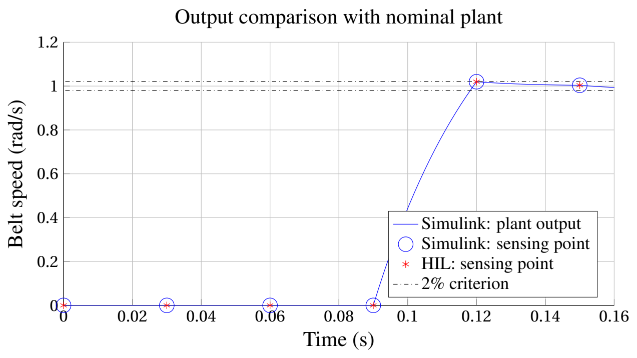


Figure 3.8: Performance comparison three-core pipelined controller.

Fig. 3.9 compares the controller performance using the three available resource configurations. Every newly added pipe improves the QoC of the system. The resource configuration with three pipes outperforms the other two, due to its higher sampling rate.

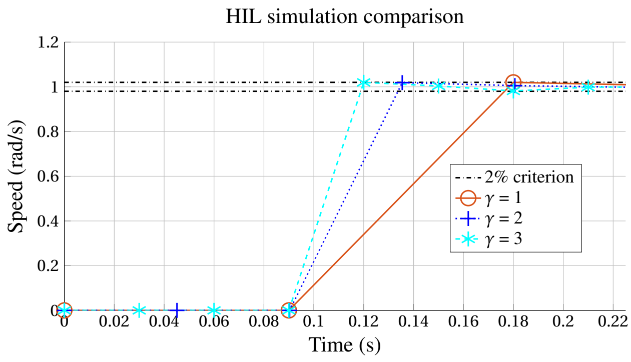


Figure 3.9: Comparison of step responses using HIL simulation.

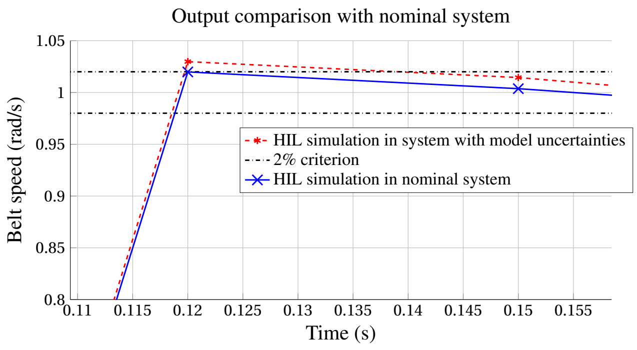


Figure 3.10: Output of HIL simulation when the system with model uncertainties is considered.

The robustness of the pipelined controller is also tested using HIL simulation. We add to the nominal model a constant uncertainty of $\Delta A_c^{(1,1)} = 1.15$. The model is discretized and implemented in the corresponding tile. The result is shown in Fig. 3.10 for $\gamma = 3$. The controller is still stable, although the settling time is increased from $S_t = 118\ ms$ in the nominal case to $S_t = 138\ ms$ in the case with model uncertainties.

3.10 Summary

In this chapter, we have presented an analysis framework to include model uncertainties in the resource-QoC trade-off in pipelined systems. The framework includes methods to either analyse the impact of models with uncertainties in performance-oriented pipelined controllers or to design robustness-oriented pipelined controllers. Our framework starts with a technique to approximate discrete-time uncertainties based on continuous-time uncertainties. Such uncertainties are presented in the form of time-invariant matrices with one uncertain element. Our results show that the robustness of a performance-oriented pipelined controller decreases with each newly added pipe, because adding sensing resources produces a more aggressive response which is more susceptible to modelling errors. Therefore, our framework may be used in a performance-oriented design to select a resource configuration that not only provides performance improvement but also guarantees that a minimum desired robustness is met. Likewise, our results show that the robustness of a pipelined controller can be increased with each newly added pipe in a robustness-oriented design. Therefore, our framework can be used to select a resource configuration that provides a desired robustness. We presented a HIL simulation to show the feasibility of pipelined control. The HIL simulation validates our analysis, results, and observations.

As future work, it would be interesting to confirm our numerical analysis to reason about the relation between continuous-time and discrete-time uncertainties with an analytical argument and to develop an analysis for a more general class of uncertainties. Further, analysing existing design methods for pipelined control may provide insight whether it is possible to design a controller that improves robustness and control performance at the same time.

Chapter 4

Reconfigurable pipelined control

4.1 Preliminaries 76
4.2 Problem definition 76
4.3 Related work 78
4.4 Modelling RPC 79
4.5 Control design in RPC 85
4.6 Summary of the design flow 98
4.7 Case study: xCPS 99
4.8 Simulation results 101
4.9 Summary 108

In pipelined-sensing control, increasing the processing resources shortens the sampling period, which can be used to improve the QoC. However, in Cyber-Physical Systems (CPSs) the processing resources are commonly shared between the DISC (e.g., a pipelined-sensing controller) and other applications. For example, in Advanced Driver Assistance Systems (ADAS), the pipelined controller may share the processing resources with other applications such as the air conditioning, the alarm system or the GPS system. Such computational applications are sporadic, i.e., their processing resources are not being continuously used and the processor occasionally runs idle. Dynamically allocating the processing resources between the pipelined-sensing controller and other sporadic applications can be used to reduce processor idling improving resource usage and the overall QoC. In this chapter, we show that the QoC of a pipelined-sensing controller can be further improved by dynamically allocating (i.e.

reconfiguring) processing resources that are temporarily unused by other applications to the sensing pipeline. To do so, we present a design strategy for a Reconfigurable Pipelined Controller (RPC), that allows for run-time reconfiguration of the sensing resources. Based on our results, we provide insights of when it is beneficial to use RPC over static pipelined control.

The contents of this chapter were published in SIES 2016 [90]. A concrete instance of the RPC framework is described in [89].

4.1 Preliminaries

Control systems: The RPC uses the linear time-invariant plant model presented in Eq. 2.1:

$$\begin{aligned}\dot{x}(t) &= A_c x(t) + B_c u(t)\\ y(t) &= C_c x(t),\end{aligned} \tag{4.1}$$

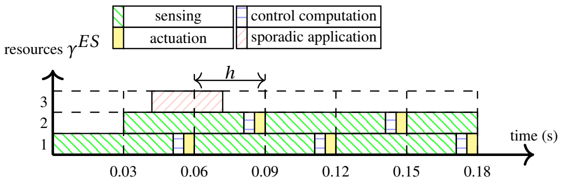
where the plant is defined by $A_c \in \mathbb{R}^{n\times n}$ the state, $B_c \in \mathbb{R}^{n\times 1}$ the input, and $C_c \in \mathbb{R}^{1\times n}$ the output matrices. $x(t) \in \mathbb{R}^{n\times 1}$ is the state vector, $u(t) \in \mathbb{R}$ the input, and $y(t) \in \mathbb{R}$ the output at time $t \in \mathbb{R}^{\geq}$. The plant has $n \in \mathbb{Z}^+$ states.

Embedded systems: We consider an embedded system with $\gamma^{ES} \in \mathbb{Z}^+$ processing resources. $\gamma^{sa} \in \mathbb{Z}^+$ processing resources are needed for $sa \in \mathbb{Z}^+$ sporadic applications. The remaining $\gamma = \gamma^{ES} - \gamma^{sa}$ processing resources can be used to design a static pipelined controller, as described in Chapter 2. However, an RPC with improved QoC can also be designed by dynamically allocating the γ processing resources available for the RPC and any available resources from γ^{sa} to the controller. An RPC therefore dynamically uses a maximum of $\gamma^{MC} \in \mathbb{Z}^+$ processing resources, with $\gamma^{ES} \geq \gamma^{MC} > \gamma$. In this chapter, we consider a data-intensive sensor with an external trigger. Following the arguments of Section 2.3.4, an external trigger is commonly available in data-intensive sensors. Such a trigger is required to enable the capturing of image data during the controller reconfiguration. Note that a periodically triggered camera can also be supported, as described in [89].

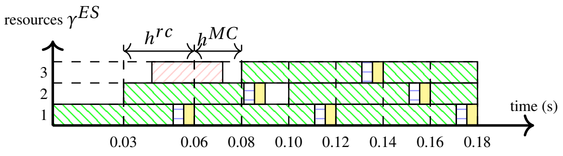
4.2 Problem definition

An RPC on-line changes (i.e. reconfigures) the number of processing resources being used by the controller. An example of an RPC is shown in Fig. 4.1. The embedded system has three processing resources that execute a pipelined controller and a sporadic application. Using the strategy of Chapter 2, a static pipelined controller can be designed with the two statically available processing resources.

However, an additional processing resource is available when the sporadic task is not executing. Therefore, an RPC composed of two controllers can be designed: one for two and one for three processing resources. The controllers in the RPC may switch depending on the availability of processing resources. The RPC potentially improves the QoC compared to the static controller with two processing resources. An RPC therefore has different resource configurations (e.g., two in Fig. 4.1b) depending on the availability of processing resources. The RPC example in Fig. 4.1b runs initially with two processing resources, leading to a sampling period h^{rc}. The controller reconfigures at some point to a configuration with three processing resources, leading to a sampling period h^{MC}. The result is a switched controller.



(a) Static pipelined control. The sampling period h remains constant during the entire operation of the controller. One processing resource is dedicated exclusively to the sporadic task.



(b) Reconfigurable pipelined control. The processing resources are dynamically allocated between the controller and the sporadic application. The sampling period changes at run-time between h^{rc} and h^{MC}.

Figure 4.1: Examples of resources configurations.

To reconfigure the processing resources in pipelined-sensing control, a modelling and a control-design strategy are required. The modelling strategy needs to capture the interplay between variable resource configurations and the plant model. Variable resource configurations produce multiple sampling periods, which change depending on the resource configuration. Therefore, the control-design strategy needs to guarantee stability with all the resulting sampling periods while providing QoC improvement compared to a static pipelined control.

Our contributions in this chapter are two fold: (i) we adapt the modelling strategy of

[24] to capture the interplay of varying processing resources with the dynamic system; (ii) this modelling strategy is used to design reconfigurable controllers that outperform the static pipelined implementation, while guaranteeing stability.

4.3 Related work

Creating a reconfigurable controller requires a modelling technique that captures the changes in the plant, and a control-design technique that guarantees stability and QoC improvement. Reconfigurable control can be found in embedded control applications (e.g., [122]) and networked control systems (e.g., [120]). Common techniques to design reconfigurable controllers include periodic linear systems, switched linear systems, and robustness analysis. None of these techniques have been used to reconfigure the processing resources in a pipelined control.

We consider control-design techniques that potentially allow reconfiguration in a pipelined-sensing controller. Periodically switched linear systems are commonly used in embedded control where multiple periodic applications share a processing resource. This resource sharing creates implementations where the controllers have multiple-sampling periods and a global hyper-period. Design strategies for periodically switching controllers include periodic pole placement [26, 129, 138, 86], periodic Lyapunov functions [138, 36, 81, 119], and periodic LQR [15, Chapter 1.8],[93, 27, 60, 128, 122]. These control-design strategies can be used to guarantee QoC improvement in terms of real-time performance metrics. Therefore, these approaches can be adapted to design reconfigurable pipelined controllers. However, the resulting RPC would be limited to applications with periodic sporadic tasks.

For switched linear systems, Common-Quadratic Lyapunov Functions (CQLFs) can be used to design a set of controllers that allows stable arbitrary switching between controllers with different sampling periods [87, 112, 95, 122]. Therefore, in switched linear systems the switching sequence is not required to be periodic. Following the ideas of [122], CQLFs allow to design a controller with optimized QoC while the rest of the controllers guarantee stability.

Robustness-analysis techniques are commonly used in networked control where the network induces variable sampling periods, variable delay, and packet losses. This technique provides controllers with robust stability guarantees on a controller given the changing network conditions [143, 25, 24, 102]. This control design can be adapted to RPC. However, the resulting robust stability guarantees are not easily related to real-time performance metrics, which is not aligned with the scope of this thesis. Ref. [24] also presents a modelling design technique used to analyse the stability of systems with variable sensing delay. We adapt such a technique for modelling reconfigurable controllers to capture the interplay of varying resources with the dynamic system.

In this chapter, we combine the modelling strategy of Chapter 2 with the ideas of [24], to include the multiple resource configurations of an RPC. Further, we use the RPC model to design a set of controllers using CQLFs. The controllers provide QoC improvement compared to a static pipelined controller design, while allowing stable arbitrary switching between multiple resource configurations.

4.4 Modelling RPC

The model of an RPC captures the interplay between the plant model of Eq. 4.1 and the allocated processing resources which dynamically change depending on their availability. To capture this variability, our RPC is composed from multiple resource configurations:

- a Maximal Configuration (MC) denoted by MC with $\gamma^{MC} \in \mathbb{Z}^+$ processing resources. The MC uses the maximum number of resources designated for the RPC, i.e., $\gamma^{MC} \leq \gamma^{ES}$. The MC has a sampling period given by $h^{MC} \in \mathbb{R}^+$.
- a set of Reduced Configurations (RCs) denoted by RC, with $|RC| < \gamma^{MC}$. Each RC $rc \in RC$ has $\gamma^{rc} \in \mathbb{Z}^+$ processing resources. The RCs use fewer processing resources than the MC (i.e., $\gamma^{rc} < \gamma^{MC}$) to release processing resources for other applications. The number of RCs is a design parameter that depends on the presence of other sporadic applications. Each RC rc has a sampling period given by $h^{rc} \in \mathbb{R}^+$.

It is assumed that all the possible configurations from the RPC are known at design time. At run-time, the RPC switches between the resource configurations (i.e., the MC and the RCs) and their corresponding sampling periods

$$\mathbb{H} = \left\{h^c \mid c \in RC \cup \{MC\}\right\}.$$

Due to the variability of sampling periods, the controller sensing instants (i.e., the moments when a sensing task starts) are unequally spaced. Therefore, to keep a notion of time in the discrete-time models, the following equation is defined:

$$t_k^s = \sum_{i=0}^{k-1} h_i, \tag{4.2}$$

with $h_i \in \mathbb{H}$ the sampling period of the RPC at sample i ($i \in \mathbb{Z}^{\geq}$) and $t_k^s \in \mathbb{R}^{\geq}$ the start of the controller sensing instant at the discrete-time index $k \in \mathbb{Z}^{\geq}$, with $t_0^s = 0$. Note that the corresponding actuation operations are completed at time $t = t_k^s + \tau$.

Example 4.1. ***RPC sensing instants:*** *Consider the resource configuration shown in Fig. 4.2. The sensing, control computation, and actuation operations result in delays of* $\tau_s = 0.05$ *s,* $\tau_c = 0.005$ *s, and* $\tau_a = 0.005$ *s, respectively. The controller is composed of an MC with* $\gamma^{MC} = 3$ *and* $h^{MC} = 0.02$ *s, and an RC with* $\gamma^{rc} = 2$ *and* $h^{rc} = 0.03$ *s. The controller switches at run time from the RC to the MC and back to the RC, which results in unevenly distributed sampling instants, sometimes being* 0.02 *s apart, sometimes* 0.03 *s. Applying Eq. 4.2, the relationship between the discrete-time index and the sampling instants is found. The resulting timings are shown in Table 4.1.*

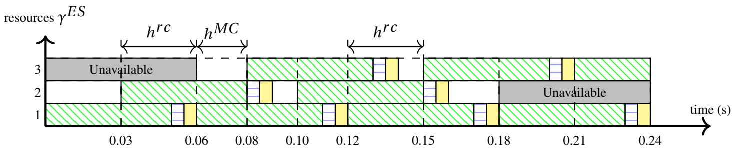


Figure 4.2: Sensing instants on an RPC. The controller switches temporarily from an RC to the MC which causes unevenly distributed sampling instants. The “unavailable” processing resources are then used by sporadic applications.

Table 4.1: Timings, state vector, and control inputs in Fig. 4.2. The state vectors and control inputs are explained in the following subsections.

k	core	sensing			control computation	
		$t_k^s(s)$	Eq. 4.2 expansion	measured state	$t_k^c(s)$	controller
0	1	0.00	t_0^s	x_0^{rc}	0.05	u_0^{rc}
1	2	0.03	$t_1^s = h^{rc}$	x_1^{rc}	0.08	u_1^{rc}
2	1	0.06	$t_2^s = t_1^s + h^{rc}$	x_2^{MC}	0.11	u_2^{MC}
3	3	0.08	$t_3^s = t_2^s + h^{MC}$	x_3^{MC}	0.13	u_3^{MC}
4	2	0.10	$t_4^s = t_3^s + h^{MC}$	x_4^{MC}	0.15	u_4^{MC}
5	1	0.12	$t_5^s = t_4^s + h^{MC}$	x_5^{rc}	0.17	u_5^{rc}
6	3	0.15	$t_6^s = t_5^s + h^{rc}$	x_6^{rc}	0.20	u_6^{rc}
7	1	0.18	$t_7^s = t_6^s + h^{rc}$	x_7^{rc}	0.23	u_7^{rc}
8	3	0.21	$t_8^s = t_7^s + h^{rc}$	x_8^{rc}	0.26	u_8^{rc}

4.4.1 Modelling the MC

The modelling of an MC follows the modelling of pipelined control of Section 2.3 and the ideas of [24]. We summarize the modelling strategy with an updated notation for RPC.

Consider a DISC with a sensing-to-actuating delay of $\tau \in \mathbb{R}^+$ seconds and $\gamma^{MC} \in \mathbb{Z}^+$ processing resources available for computation. The sampling period of the MC h^{MC} is:

$$h^{MC} = \frac{\tau}{\gamma^{MC}}. \tag{4.3}$$

Discretizing Eq. 4.1 with the sampling period h^{MC} and sensing-to-actuating delay τ leads to the following discrete-time representation:

$$\begin{aligned} x_{k+1}^{MC} &= A^{MC} x_k^{MC} + B^{MC} u_{k-\gamma^{MC}}^{MC} \\ y_k^{MC} &= C_c x_k^{MC}, \end{aligned} \tag{4.4}$$

where $x_k^{MC} \in \mathbb{R}^{n\times 1}$ and $y_k^{MC} \in \mathbb{R}$ are the discrete-time state vector and output of the MC, respectively, with $x_k^{MC} := x(t_k^s)$ and $y_k^{MC} := y(t_k^s)$ when $h_{k-\gamma^{MC}} = h^{MC}$. $u_{k-\gamma^{MC}}^{MC} \in \mathbb{R}$ is the delayed controller input for the MC, which is implemented using a ZOH to keep the actuation signal constant between consecutive samples i.e., $u(t) := u_{k-\gamma^{MC}}^{MC}$ for $t \in [t_k^s, t_{k+1}^s)$. C_c is defined in Eq. 4.1 while $A^{MC} \in \mathbb{R}^{n\times n}$ and $B^{MC} \in \mathbb{R}^{n\times 1}$ are the discrete-time state and input matrices defined by [8]:

$$\begin{aligned} A^{MC} &= e^{A_c h^{MC}} \\ B^{MC} &= \int_0^{h^{MC}} e^{A_c s} B_c ds. \end{aligned} \tag{4.5}$$

To remove the sensing-to-actuating delay from Eq. 4.4, the augmented notation for the MC is defined as:

$$z_k^{MC} = \left[\left(x_k^{MC}\right)^T \quad u_{k-\gamma^{MC}}^{MC} \quad u_{k-\gamma^{MC}+1}^{MC} \quad \cdots \quad u_{k-1}^{MC} \right]^T. \tag{4.6}$$

with $z_k^{MC} \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$ the discrete-time augmented state vector. Reorganizing Eq. 4.4 gives the following augmented system:

$$\begin{aligned} z_{k+1}^{MC} &= \Phi^{MC} z_k^{MC} + \Gamma^{MC} u_k^{MC} \\ y_k^{MC} &= C_d z_k^{MC}, \end{aligned} \tag{4.7}$$

with $\Phi^{MC} \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$, $\Gamma^{MC} \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$, and $C_d \in \mathbb{R}^{1\times(n+\gamma)}$ the augmented discrete-time state, input and output matrices respectively, further defined by:

$$\begin{aligned}\Phi^{MC} &= \begin{bmatrix} A^{MC} & B^{MC} & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \\ \Gamma^{MC} &= \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}^T \\ C_d &= \begin{bmatrix} C_c & 0 & 0 & \dots & 0 \end{bmatrix},\end{aligned} \tag{4.8}$$

where 0 denotes zero matrices of appropriate dimensions.

In the augmented-state-vector definition of Eq. 4.6, the number of states depends on γ^{MC}. Similarly, in a particular RC, the number of states depends on a particular γ^{rc}, which leads to a discrete-time model with fewer states than the MC. However, in order to design an RPC, it is necessary that the discrete-time models of both the MC and each RC have an equal number of states. In the next section, we propose a modelling strategy for the RCs that addresses this aspect.

Example 4.2. ***MC modelling:*** *Consider the dynamic system presented in Section 2.2:*

$$A_c = \begin{bmatrix} 0 & 1.7 \\ -9 & -2.5 \end{bmatrix}, B_c = \begin{bmatrix} 0 \\ 10 \end{bmatrix}, C_c = [1\ 0].$$

A camera and an image-processing algorithm are used as a sensor for measuring the plant states x^1 and x^2. The total sensing-to-actuating delay is $\tau = 0.084$ s. An embedded platform with two processing resources is available for the DISC and one sporadic application. Therefore, we select $\gamma^{ES} = \gamma^{MC} = 2$.

4

Using Eq. 4.3, the sampling period is defined as $h^{MC} = 42$ ms. Using Eq. 4.5, the discretized model is given by:

$$A^{MC} = \begin{bmatrix} 0.98 & 0.07 \\ -0.35 & 0.88 \end{bmatrix}, B^{MC} = \begin{bmatrix} 0.01 \\ 0.39 \end{bmatrix}.$$

The model augmentation of Eqs. 4.6 and 4.8 gives:

$$z_k^{MC} = \begin{bmatrix} x_k^{MC,1} \\ x_k^{MC,2} \\ u_{k-2}^{MC} \\ u_{k-1}^{MC} \end{bmatrix}, \Phi^{MC} = \begin{bmatrix} 0.98 & 0.07 & 0.01 & 0 \\ -0.35 & 0.88 & 0.39 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \Gamma^{MC} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.9}$$

4.4.2 Modelling the RCs

Consider a resource configuration $rc \in RC$. The sampling period of such a configuration is defined by:

$$h^{rc} = \frac{\tau}{\gamma^{rc}}. \tag{4.10}$$

Discretizing Eq. 4.1 with the sampling period h^{rc} and sensing-to-actuating delay τ leads to the following discrete-time representation:

$$\begin{aligned} x^{rc}_{k+1} &= A^{rc} x^{rc}_k + B^{rc} u^{rc}_{k-\gamma^{rc}} \\ y^{rc}_k &= C_c x^{rc}_k, \end{aligned} \tag{4.11}$$

where $x^{rc}_k \in \mathbb{R}^{n\times 1}$ and $y^{rc}_k \in \mathbb{R}$ are the discrete-time state vector and output of the RC, respectively, with $x^{rc}_k := x(t^s_k)$ and $y^{rc}_k := y(t^s_k)$ and $h_{k-\gamma^{rc}} = h^{rc}$. $u^{rc}_{k-\gamma^{rc}} \in \mathbb{R}$ is the controller input for the RC, which is implemented using a ZOH to keep the actuation signal constant between consecutive samples i.e., $u(t) := u^{rc}_{k-\gamma^{rc}}$ for $t \in [t^s_k, t^s_{k+1})$. C_c is defined in Eq. 4.1 while $A^{rc} \in \mathbb{R}^{n\times n}$ and $B^{rc} \in \mathbb{R}^{n\times 1}$ are the discrete-time state and input matrices, further defined by:

$$\begin{aligned} A^{rc} &= e^{A_c h^{rc}}, \\ B^{rc} &= \int_0^{h^{rc}} e^{A_c s} B_c ds. \end{aligned} \tag{4.12}$$

To remove the actuation delay from Eq. 4.11, a strategy similar to the MC case could be used:

$$\tilde{z}^{rc}_k = \begin{bmatrix} \left(x^{rc}_k\right)^T & u^{rc}_{k-\gamma^{rc}} & u^{rc}_{k-\gamma^{rc}+1} & \cdots & u^{rc}_{k-2} & u^{rc}_{k-1} \end{bmatrix}^T, \tag{4.13}$$

with $\tilde{z}^{rc}_k \in \mathbb{R}^{(n+\gamma^{rc})\times 1}$. However, since $\gamma^{rc} < \gamma^{MC}$, the augmented state vector in Eq. 4.6 has a larger dimension than the augmented state vector in Eq. 4.13. We make the augmented state-vector dimension in all the configurations identical to ease the stability analysis of the overall system. In order to equalize the number of states in all configurations, Eq. 4.13 is augmented by introducing old controller inputs u^{rc}_k in the augmented state-vector definition, so that the length of z^{rc}_k is the same as z^{MC}_k:

$$z^{rc}_k = \begin{bmatrix} \left(x^{rc}_k\right)^T & u^{rc}_{k-\gamma^{MC}} & u^{rc}_{k-\gamma^{MC}+1} & \cdots & u^{rc}_{k-\gamma^{rc}} & \cdots & u^{rc}_{k-1} \end{bmatrix}^T \tag{4.14}$$

with $z^{rc}_k \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$ the discrete-time augmented state vector. The discrete-time augmented model of the RC is then defined by:

$$\begin{aligned} z^{rc}_{k+1} &= \Phi^{rc} z^{rc}_k + \Gamma^{rc} u^{rc}_k \\ y^{rc}_k &= C_d z^{rc}_k, \end{aligned} \tag{4.15}$$

where C_d is defined in Eq. 4.8, and $\Phi^{rc} \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$ and $\Gamma^{rc} \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$ are the discrete-time state and input matrices of the rc, respectively, defined by:

$$\Phi^{rc} = \left[\begin{array}{ccccccc} & & & \multicolumn{4}{c}{\overbrace{\qquad\qquad\qquad\qquad}^{\gamma^{rc}}} \\ A^{rc} & 0 & \dots & B^{rc} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ & \multicolumn{6}{c}{\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\gamma^{MC}}} \end{array}\right], \Gamma^{rc} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}. \tag{4.16}$$

Note that there are added columns and rows of zeros in these matrices (e.g., between A^{rc} and B^{rc} in Φ^{rc}). These zeros cancel the effect of some controller inputs in the plant dynamics. The cancelled controller inputs correspond to the ones added in the augmented state definition of Eq. 4.14.

Example 4.3. ***RC modelling:*** *Continuing with Example 4.2, we select to have one reduced configuration with $\gamma^{rc_1} = 1$. Using Eq. 4.10, the sampling period is given by:*

$$h^{rc_1} = 0.084\ s.$$

Replacing h^{rc_1} in Eq. 4.12 gives the discrete-time model:

$$A^{rc_1} = \begin{bmatrix} 0.95 & 0.12 \\ -0.66 & 0.76 \end{bmatrix}, B^{rc_1} = \begin{bmatrix} 0.05 \\ 0.74 \end{bmatrix}.$$

The discrete-time augmented vector is defined using Eq. 4.14 as:

$$z_k^{rc_1} = \begin{bmatrix} x_k^{rc_1,1} & x_k^{rc_1,2} & u_{k-2}^{rc_1} & u_{k-1}^{rc_1} \end{bmatrix}^T.$$

The augmented state and input matrices are given by:

$$\Phi^{rc_1} = \begin{bmatrix} 0.95 & 0.12 & 0 & 0.05 \\ -0.66 & 0.76 & 0 & 0.74 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \Gamma^{rc_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

The models of the MC (Eqs. 4.6-4.8) and of the RC (Eqs. 4.14-4.16) are used to design an RPC in the next section.

4.5 Control design in RPC

Depending on the number of available processing resources, the active configuration in the RPC switches between the MC and one or more RCs. Such a switching strategy can potentially lead to an unstable closed-loop behaviour [117]. The control-design strategy has to cope with this problem. Additionally, since the MC has the largest number of processing resources, it can achieve a higher QoC. The RPC is then designed to operate most of the time in the MC, while it might occasionally switch to an RC to release processing resources for sporadic applications. Therefore, the control-design strategy guarantees QoC while running in the maximal configuration, and stability while switching and running in an RC. The purpose of this section is

- to present a set of requirements for the embedded system (e.g., activation and de-activation policies) and the control system (state initialization) that enable the switching mechanism;
- to design feedback gains K^{MC} and K^{rc} (for each rc) that allow performance improvement and stable switching, respectively, as is shown in Fig. 4.3;
- to design feed-forward gains F^{MC} and F^{rc} (for each rc) such that the output vector y converges to the input step reference r, when one of the configurations is active for a sufficiently long time.

The last point is called set-point regulation. Although set-point regulation in a switched system is challenging when the active system changes fast, it is feasible in RPC because the controller is expected to stay primarily in only one configuration (i.e., the MC) [29]. In line with the arguments of Section 2.4.1.4, the controllers are designed in two steps i.e., first the feedback gains and then the feed-forward gains. This allows us to treat stability and performance aspects separately. Alternative design strategies such as LQI can also be used to design both gains in one step. In principle, these approaches can be adapted to our work. However, for a switched-system design, it is hard to find controllers that improve QoC in terms of time-domain metrics for LQI formulations [94]. This motivates us to use the current two-step approach.

The control law has the following forms:

$$u_k^{MC} = K^{MC} z_k^{MC} + F_k^{MC} r \tag{4.17}$$

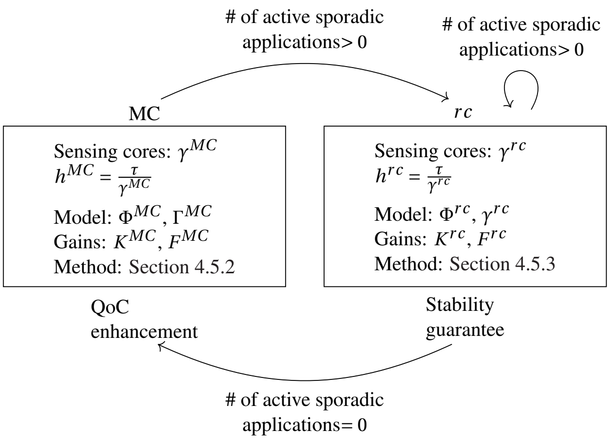

# of active sporadic applications> 0
# of active sporadic applications> 0
MC
rc
Sensing cores: γ^{MC}
$h^{MC} = \frac{\tau}{\gamma^{MC}}$
Model: Φ^{MC}, Γ^{MC}
Gains: K^{MC}, F^{MC}
Method: Section 4.5.2
Sensing cores: γ^{rc}
$h^{rc} = \frac{\tau}{\gamma^{rc}}$
Model: Φ^{rc}, γ^{rc}
Gains: K^{rc}, F^{rc}
Method: Section 4.5.3
QoC enhancement
Stability guarantee
# of active sporadic applications= 0

Figure 4.3: Operation modes of an RPC. One of the RCs becomes active when one or multiple sporadic applications are triggered. Otherwise the MC becomes active.

$$u_k^{rc} = K^{rc} z_k^{rc} + F_k^{rc} r, \tag{4.18}$$

where $K^{MC} \in \mathbb{R}^{1\times(n+\gamma^{MC})}$, $K^{rc} \in \mathbb{R}^{1\times(n+\gamma^{MC})}$ and $F^{MC} \in \mathbb{R}$, $F^{rc} \in \mathbb{R}$ are the feedback and feed-forward gains, respectively. Note that when the controller starts with the MC, we apply $u_k^{MC} = 0$ for $k < 0$ since we need to wait for the first sample to compute the control input. Likewise, when the controller starts with an RC, $u_k^{rc} = 0$ for $k < 0$.

In the following subsection, the requirements of the switching mechanism are elaborated. Then, a controller design strategy is first provided for the MC in Section 4.5.2, where controller performance is enhanced, and then for the RC in Section 4.5.3, where the stability is guaranteed while switching occurs.

4.5.1 Switching mechanism

Switching between multiple resource configurations requires the definition of a run-time switching mechanism in the embedded system and in the control system. This guarantees that smooth transitions are achieved between the changing resource configurations. The switching mechanism covers the following items, with related requirements:

- The (delayed) activation and deactivation policies of the processing resources used for running control operations, considering possible unavailability of the processing resources. For example, consider the RPC of Fig. 4.2. A new sensing operation starts (i.e., activation policy) at processing resource three at time $t = 0.08\ s$, while the next sensing operation is delayed (i.e., delayed activation policy) on processing resource two from $t = 0.09\ s$ to $t = 0.10\ s$. These policies ensure a smooth transition in the sampling periods. Later during the execution, processing resource two becomes unavailable because it is needed for the sporadic task. The activation policies need to take this into account.

- Augmented state-vector initialization during the switching. This state initialization ensures that the control laws from Eqs. 4.17 and 4.18 can be computed during the switching. For example, in the RPC of Fig. 4.2, to compute the control law u_2^{MC} at $t = 0.11\ s$ (see Table 4.1), the augmented state vector z_2^{MC} is required. This augmented state vector is composed of x_2^{MC} and the old control inputs of the MC u_1^{MC}, u_0^{MC}, and u_{-1}^{MC} (see Eq. 4.6). x_2^{MC} was measured by the sensing operation started at $t = 0.06\ s$. However, the old controller inputs were never computed because the active configuration at $t \leq 0.06\ s$ was an RC. Therefore, z_2^{MC} needs to be initialized.

In the following subsections, we introduce some general rules and definitions for the reconfiguration mechanism. Then, since the transition from the MC to an RC results in other (de-)activation policies and state initialization than the transition from an RC to the MC, we analyse these transitions in independent subsections. Notice that in RPC, also RC-to-RC reconfigurations are possible. A reconfiguration to less processing resources follows the mechanism of the MC to an RC transition. Likewise, a reconfiguration to more processing resources follows the mechanism of an RC to the MC transition. Therefore, we do not discuss those RC-to-RC reconfigurations separately.

4.5.1.1 General rules and definitions for the switching mechanism

We consider the following key terms and definitions for the reconfiguration mechanism.

1. One of the processing resources in the RPC is referred to as the *base* processing resource during the reconfiguration. This processing resource controls the reconfiguration to ensure smooth timings during the controller reconfiguration. The activation of the base processing resource is not adjusted during the switching. When the sensing operation starts in the base processing resource, then the new sampling period of the RPC becomes active. For example in Fig. 4.2, core one corresponds to the base processing resource during both reconfigurations.

Notice that the sampling periods always change on the activation of this processing resource. The RPC needs to be implemented in such a way that any of the allocated processing resources can act as base processing resource.

2. The rest of the processing resources are referred to as *non-base* processing resources during the reconfiguration. These processing resources are at run-time reconfigured between the RPC and any of the sporadic tasks. The non-base processing resources are programmed to run either the RPC or the sporadic tasks. For example in Fig. 4.2, core two and three correspond to non-base processing resources during both reconfigurations.

3. Each sporadic task is enabled to run on any of the processing resources. This is required because during reconfigurations, the RPC might release other processing resources than the originally available ones (see for example in Fig. 4.2, where the processing resource tagged as "unavailable" changes from three to two after two reconfigurations).

4. The switching occurs between two time instants: a switching initiation $t^s_{k_{sw}} \in \mathbb{R}^+$ (with $k_{sw} \in \mathbb{Z}^{\geq}$ the discrete-time index at the switching initiation) and a switching completion. The switching initiation starts when the sampling period changes between configurations. The switching completion occurs τ time units after the switching initiation, i.e., at time $t^s_{k_{sw}} + \tau$. For example in Fig. 4.2, the first reconfiguration starts at $t^s_2 = 0.06\ s$, with $k_{sw_1} = 2$, and completes at $t = 0.12\ s$; the second reconfiguration then immediately starts at $t^s_5 = 0.12\ s$, with $k_{sw_2} = 5$, and completes at $t = 0.18\ s$.

5. Processing resources can only be assigned to sporadic tasks from a switching completion onwards. Therefore, the start time of a sporadic task needs to be communicated to the RPC τ seconds before the desired switching competition. For example in Fig. 4.2, processing resource two is made unavailable to the RPC (i.e., there is a desired switching completion) from time $t = 0.18\ s$ onwards. That implies that the RPC is aware of such unavailability from $t < 0.12\ s$, which allows it to orchestrate the reconfiguration from $t^s_5 = 0.12\ s$.

6. New processing resources can only be assigned to the RPC from a switching initiation onwards. Therefore, to keep the processing resources with the highest utilization, the completion time of a sporadic task needs to be communicated to the RPC before the desired switching initiation. For example in Fig. 4.2, processing resource three is made available to the RPC at $t = 0.06\ s$. That implies that the RPC is aware of such a completion at some $t < 0.06\ s$, such that t^s_2 is used as

a switching initiation. Note that in case the completion time is not communicated to the RPC, then the controller can only start to orchestrate the reconfiguration when it notices that the processing resource is available (e.g., from $t > 0.06$ s), leading to a lower resource utilization.

4.5.1.2 Switching mechanism: from the MC to an RC

Consider the example shown in Fig. 4.4. The processing resources are numbered from 1 to γ^{MC}. The switching initiation starts when the sampling period switches from h^{MC} to the period h^{rc} of an RC. That is, $h_{k_{sw}-1} = h^{MC}$ and $h_{k_{sw}} = h^{rc}$. $t^s_{k_{sw}}$ is then formally defined as the time instant resulting from substituting k_{sw} for k in Eq. 4.2. We define activation (or deactivation) policies and state-initialization policies around this time instant.

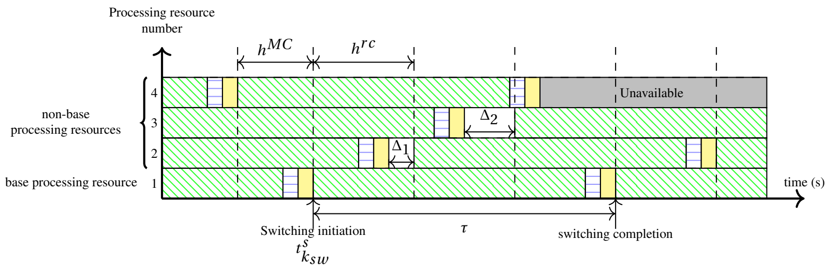


Figure 4.4: Example of switching from an MC with $\gamma^{MC} = 4$ to an RC with $\gamma^{rc} = 3$, where $\Delta_1 = \Delta_1^{rc} = h^{rc} - h^{MC}$ and $\Delta_2 = \Delta_2^{rc} = 2(h^{rc} - h^{MC})$.

1. **Delayed activation and deactivation policies in MC-to-RC transition:** A smooth transition from h^{MC} to h^{rc} is achieved by delaying the activation or deactivation of the processing resources at precise time instants. Fig. 4.4 shows an example of the switching mechanism between the MC and an RC. We establish the following transition requirements.

   (a) $\gamma^{MC} - \gamma^{rc}$ non-base processing resources are made unavailable to the RPC (i.e., they are deactivated). The processing resources originally scheduled to start a sensing operation at time

$$t^{s,MC}_{k,j} = t^s_{k_{sw}} + \tau - jh^{MC},$$

with $j = \{1, \ldots, \gamma^{MC} - \gamma^{rc}\}$, are deactivated for the RPC. These processing resources are deactivated as soon as they finish their ongoing sensing, control computation, and actuation operations. These processing resources are then made available to run any of the sporadic tasks from the switching completion onwards. For example in Fig. 4.2, processing resource two was originally scheduled to start a sensing operation at time $t^{s,MC}_{5,1} = 0.16$ s. However, due the reconfiguration, this processing resource is made unavailable to the RPC.

(b) The start time of the sensing task of the remaining non-base processing resources is delayed from:

$$t^{s,MC}_{k,i} = t^{s}_{k_{sw}} + ih^{MC} \tag{4.19}$$

to

$$t^{s,rc}_{k,i} = t^{s}_{k_{sw}} + ih^{MC} + \Delta^{rc}_{i}, \tag{4.20}$$

with $i = \{1, ..., \gamma^{rc} - 1\}$. $\Delta^{rc}_{i} = i(h^{rc} - h^{MC})$ is a delay on the activation time. This activation delay guarantees a smooth transition in the sampling periods. For example in Fig. 4.2, the start time of the sensing task in processing resource three is delayed from $t^{s,MC}_{5,1} = 0.14$ s to $t^{s,rc}_{5,1} = 0.15$ s, with $\Delta^{rc}_{1} = 0.01$ s.

(c) Switching from any configuration $\gamma^{MC} > 1$ to $\gamma^{rc} = 1$ does not need any adjustment in the activation mechanism, because only the base processing resource remains running after the switch.

(d) Due to the sensing delay, the change in the sampling period is only reflected on the actuation side (i.e., the actuation period) τ time units after the switching initiation. Consequently, the controller actuation period still follows h^{MC} between the switching initiation and the last sample before the switching completion, i.e., during the time frame $[t^{s}_{k_{sw}}, t^{s}_{k_{sw}} + \tau)$. This implies that the control law of the RC is only applied from the switching completion onwards. For example in Fig. 4.2, the sampling period changes to h^{rc} at $t = 0.12$ s. However, u^{MC}_{k} is still computed at $t = \{0.13, 0.15\}$ s. The new control law u^{rc}_{k} is computed from $t = 0.17$ s onwards.

4

2. **Augmented-state-vector initialization in MC-to-RC transition:** The control-computation operation at $t_c = t^{s}_{k_{sw}} + \tau_s$ (just before the switching completion) is prepared to compute $u^{rc}_{k_{sw}}$ using Eq. 4.18. To do so, the augmented state vector $z^{rc}_{k_{sw}}$ has to be available. Recall the augmented state definition of an RC:

$$z^{rc}_{k} = \begin{bmatrix} \left(x^{rc}_{k}\right)^{T} & u^{rc}_{k-\gamma^{MC}} & u^{rc}_{k-\gamma^{MC}+1} & \cdots & u^{rc}_{k-\gamma^{rc}} & \cdots & u^{rc}_{k-1} \end{bmatrix}^{T}. \tag{4.21}$$

$z^{rc}_{k_{sw}}$ is then composed of $x^{rc}_{k_{sw}}$ and several old control inputs $u^{rc}_{k_{sw}-i}$. The states $x^{rc}_{k_{sw}}$ were just measured by the base processing resource. However, the old control inputs $u^{rc}_{k_{sw}-i}$ are not available because the RC was inactive. Moreover, these old control inputs cannot be computed using Eq. 4.18, because they require state vectors that were not measured due to the RC being inactive. These states need to be estimated from the available information and signals. We assume that the old states are stored and available for the control-computation operation. We determine the old control inputs of $z^{rc}_{k_{sw}}$ from the available old control inputs of $z^{MC}_{k_{sw}}$. To do so, recall the augmented state-vector definition of the MC:

$$z^{MC}_{k} = \begin{bmatrix} \left(x^{MC}_{k}\right)^{T} & u^{MC}_{k-\gamma^{MC}} & u^{MC}_{k-\gamma^{MC}+1} & \cdots & u^{MC}_{k-1} \end{bmatrix}^{T}. \tag{4.22}$$

Note that the control inputs correspond to variables stored in the embedded system rather than physical states being measured (unlike x_k). Therefore, the old control inputs in the RC are initialized as:

$$u^{rc}_{k_{sw}-i} = u^{MC}_{k_{sw}-i}, \tag{4.23}$$

with $i = 1, \dots, \gamma^{rc}, \dots, \gamma^{MC}$. This initialization is a design choice that ensures that no intermediate state vectors appear during the switching. Other initializations are also possible since these are not measured states.

Note that at the switching initiation, the measured state $x^{rc}_{k_{sw}}$ would have been labelled as $x^{MC}_{k_{sw}}$ if no reconfiguration was triggered i.e., $x^{rc}_{k_{sw}} = x^{MC}_{k_{sw}}$. Therefore, the augmented state-vector initialization can be summarized as:

$$z^{rc}_{k_{sw}} = z^{MC}_{k_{sw}}. \tag{4.24}$$

This initialization facilitates the control design that is presented in Section 4.5.3, because no intermediate state vectors appear during the switching. The state initialization technique guarantees that $z^{rc}_{k_{sw}}$ is available at $t = t^{s}_{k_{sw}} + \tau_s$. Switching the control law from Eq. 4.17 to Eq. 4.18 is therefore realisable.

Example 4.4. ***Augmented-state-vector initialization in MC-to-RC transition:*** *Consider the example in Fig. 4.2. The control computation u^{rc}_{5} of $t^{c}_{5} = 0.17$ s requires the state x^{rc}_{5} and the old control inputs u^{rc}_{4}, u^{rc}_{3}, and u^{rc}_{2}, theoretically computed at times* 0.14, 0.11, *and* 0.08 *s, respectively. From Table 4.1, it is clear that x^{rc}_{5} was measured while the old control inputs are not available because the RC was not active.*

To attempt to compute the old control inputs using the control law of Eq. 4.18, consider u_4^{rc} which should have been computed at time $t_4^c = 0.14$ s. This control input depends on the physical state x_4^{rc}, which should have been measured at time $t_4^s = 0.09$ s (i.e., $t_4^s = t_4^c - \tau_s$). From Table 4.1, it is clear that no state was measured at that time. The old control inputs are therefore not computable.

Initializing the old control inputs using Eq. 4.23 yields $u_4^{rc} = u_4^{MC}$, $u_3^{rc} = u_3^{MC}$ and $u_2^{rc} = u_2^{MC}$. Table 4.1 shows that these control inputs of the MC were all previously computed. The state initialization is therefore realizable.

4.5.1.3 Switching mechanism: from an RC to the MC

Fig. 4.5 shows an example of the switching mechanism from an RC to the MC. The switching initiation starts when the sampling period switches from h^{rc} to h^{MC}. That is, $h_{k_{sw}-1} = h^{rc}$ and $h_{k_{sw}} = h^{MC}$. The switching initiation $t_{k_{sw}}^s$ is then formally defined as the time instant resulting from substituting k_{sw} for k in the notation of Eq. 4.2. We define activation policies and state-initialization policies around the switching initiation.

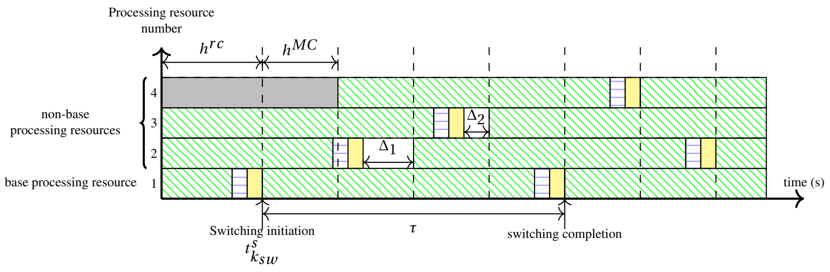


Figure 4.5: Example of switching from an RC with $\gamma^{rc} = 3$ to the MC with $\gamma^{MC} = 4$, where $\Delta_1 = \Delta_1^{MC} = 2h^{MC} - h^{rc}$, and $\Delta_2 = \Delta_2^{MC} = 3h^{MC} - 2h^{rc}$.

1. **Delayed activation and activation policy in RC-to-MC transition:** A smooth transition from h^{rc} to h^{MC} is achieved by proper activation of the processing resources at precise time instants. Fig. 4.5 illustrates the switching mechanism from an RC to the MC. Processing resources are denoted by processing resource 1 to processing resource γ^{MC}. In this case, (γ^{MC} - γ^{rc}) new processing resources are activated. We establish the following transition requirements.

(a) $\gamma^{MC} - \gamma^{rc}$ non-base processing resources are made available (i.e., activated) to the RPC between the switching initiation and completion. These new processing resources are activated at time

$$t^{s,MC}_{k,j} = t^{s}_{k_{sw}} + jh^{MC}, \tag{4.25}$$

with $j = \{1,\dots\gamma^{MC} - \gamma^{rc}\}$. For example in Fig. 4.2, processing resource three is activated at time $t^{s,MC}_{3,1} = 0.06 + 0.02 = 0.08\ s$.

(b) The activation of the non-base processing resources is delayed from:

$$t^{s,rc}_{k,l} = t^{s}_{k_{sw}} + lh^{rc} \tag{4.26}$$

to:

$$t^{s,MC}_{k,l} = t^{s}_{k_{sw}} + lh^{rc} + \Delta^{MC}_{l}, \tag{4.27}$$

with $l = 1,\dots,\gamma^{rc} - 1$ and $\Delta^{MC}_{l} = (\gamma^{MC} - \gamma^{rc})h^{MC} + lh^{MC} - lh^{rc}$ a delay in the activation time. This delayed activation time can be intuitively explained by substituting Δ^{MC}_{l} in Eq. 4.27:

$$t^{s,MC}_{k,l} = t^{s}_{k_{sw}} + (\gamma^{MC} - \gamma^{rc})h^{MC} + lh^{MC}.$$

This delayed-activation equation represents an offset produced by $\gamma^{MC} - \gamma^{rc}$ new pipes, followed by γ^{rc} delayed already active pipes activated h^{MC} time units apart. For example in Fig. 4.2, the activation of processing resource two was delayed from $t^{s,rc}_{4,1} = 0.09\ s$ to $t^{s,MC}_{4,1} = 0.10\ s$ with $\Delta^{MC}_{1} = 0.01\ s$.

(c) Switching from $\gamma^{rc} = 1$ to $\gamma^{MC} > 1$ only needs activation of new processing resources, because only the base processing resource was active before the switching initiation.

(d) Due to the sensing delay, the change in the sampling period is only reflected on the actuation side (i.e., the actuation period) τ time units after the switching initiation. Consequently, the controller actuation period still follows h^{rc} between the switching initiation and one sample before the switching completion, i.e., $t = [t^{s}_{k_{sw}}, t^{s}_{k_{sw}} + \tau)$. This implies that the control law of the MC is only applied from the switching completion onwards. For example in Fig. 4.2, the sampling period changes to h^{MC} at $t = 0.06\ s$. However, u^{rc}_{k} is still computed at $t = 0.08\ s$. The new control law u^{MC}_{k} is computed from $t = 0.11\ s$ onwards.

2. **Augmented-state-vector initialization in RC-to-MC transition:** The control-computation operation at $t_c = t^s_{k_{sw}} + \tau_s$ (just before the switching completion) is prepared to compute $u^{MC}_{k_{sw}}$ using Eq. 4.17, which further requires the augmented state vector $z^{MC}_{k_{sw}}$. Recall that $z^{MC}_{k_{sw}}$ is composed of $x^{MC}_{k_{sw}}$ and several $u^{MC}_{k_{sw}-i}$. The states $x^{MC}_{k_{sw}}$ were just measured by the base processing resource. However, the states $u^{MC}_{k_{sw}-i}$ are not available because they depend on the MC, which was inactive. Moreover these old control inputs cannot be computed using Eq. 4.17, because they require augmented state vectors of which the plant states were not measured in the RC. Therefore, these states need to be estimated from the available information and signals. We determine the elements of the augmented state vector $z^{MC}_{k_{sw}}$ from the augmented state vector $z^{rc}_{k_{sw}}$. We assume that the old states are stored and available for the control-computation operation.

   To estimate the old control inputs, we apply the same strategy as in the previous transition. Consider the state definitions presented in Eqs. 4.21 and 4.22. Note that the control inputs correspond to variables stored in the embedded system rather than physical states being measured (unlike x_k). Therefore, the old control inputs are initialized as:

$$u^{MC}_{k_{sw}-i} = u^{rc}_{k_{sw}-i}, \tag{4.28}$$

   with $i = 1, \ldots, \gamma^{MC}$. Like in the previous transition, the augmented state-vector initialization is summarized as:

$$z^{MC}_{k_{sw}} = z^{rc}_{k_{sw}}. \tag{4.29}$$

   This initialization facilitates the control design that is presented in Section 4.5.3 because no intermediate state vectors appear during the switching.

Example 4.5. ***Augmented-state-vector initialization in RC-to-MC transition:*** *Consider the RPC of Fig. 4.2. the control computation u^{MC}_2 of $t^c_2 = 0.11$ s requires the state x^{MC}_2 and the old control inputs u^{MC}_1, u^{MC}_0, and u^{MC}_{-1} computed at times* 0.09, 0.07, *and* 0.05 *s, respectively. From Table 4.1, it is clear that x^{MC}_2 was measured while the old control inputs are not available because the MC was not active. Therefore, these states need to be estimated from the available information and signals.*

To attempt to compute the old control inputs using the control law of Eq. 4.17, consider u^{MC}_1 which should have been computed at time $t^c_1 = 0.09$ s. Such an old control input depends on the physical state x^{MC}_1, which should have been measured at time $t^s_1 = 0.04$ s (i.e., $t^s_1 = t^c_1 - \tau_s$). From Table 4.1, it is clear that no state was measured at that time.

Initializing the old control inputs using Eq. 4.28 yields $u_1^{MC} = u_1^{rc}$, $u_0^{MC} = u_0^{rc}$ and $u_{-1}^{MC} = u_{-1}^{rc}$. Table 4.1 shows that these control inputs of the reduced configuration were all previously computed, except for u_{-1}^{rc}, which is assumed to be zero during the initialization. The state initialization is therefore realizable.

4.5.2 Control design in the MC

As already explained in Section 4.4, the MC has the largest number of processing resources in the RPC γ^{MC}. Therefore, the MC has the shortest sampling period in the RPC (see Eq. 4.3). Consequently, the best QoC can be achieved by the MC. The RPC is therefore expected to operate in the MC most of the time to achieve performance improvement, while occasionally it may temporarily run under an RC to release processing resources. To this end, the feedback gain K^{MC} in Eq. 4.17 is designed to enhance QoC using the design strategy presented in Section 2.2. Such strategy uses PSO to find an LQR controller with minimum settling times (i.e., maximum QoC). Likewise, the feed-forward gain F^{MC} is computed using Proposition 4.1 below, which is equal to Proposition 2.1 with updated notation of the MC.

The control law of the MC enhances the QoC as long as the RPC is running under the MC. However, due to the triggering of one or multiple sporadic applications, the system switches to an RC as illustrated in Fig. 4.3.

Proposition 4.1. *(copied from Proposition 2.1) The plant output y_k of the system of Eq. 4.7 with the control law of Eq. 4.17 converges to a constant reference r if the closed-loop system matrix $\Phi_{cl}^{MC} = \Phi^{MC} + \Gamma^{MC} K^{MC}$ is stable, the matrix $\begin{bmatrix} \Phi^{MC} - I & \Gamma^{MC} \\ C_d & 0 \end{bmatrix}$ is invertible, and F^{MC} is defined by*

$$F^{MC} = \begin{bmatrix} K^{MC} & I \end{bmatrix} \begin{bmatrix} \Phi^{MC} - I & \Gamma^{MC} \\ C_d & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

with I and 0 denoting identity and zero matrices of appropriate dimensions, respectively.

Example 4.6. ***MC controller design:*** *Continuing with Example 4.2, an LQR is tuned with the PSO algorithm of Section 2.2. The resulting feedback gain is:*

$$K^{MC} = \begin{bmatrix} -32.56 & -7.39 & -2.75 & -1.98 \end{bmatrix}.$$

Applying Proposition 4.1, the feed-forward gain is:

$$F^{MC} = 37.72.$$

The controller reaches the reference within (i.e., has a settling time of) $S_t = 153\ ms$.

4.5.3 Control design in RC

The controllers for the RCs are designed to guarantee stability if *arbitrary* switching occurs between the MC and an RC, or between RCs. To do so, consider the control law of Eq. 4.18. The feed-forward gains F^{rc} are individually found for each RC using Proposition 4.2, which is equal to Proposition 2.1 with updated notation of an RC. The feedback gains K^{rc} are simultaneously found solving the LMIs in Theorem 4.1 below, including all possible RCs and the already-computed feedback gain of the MC K^{MC}. The theorem is based on Lemmas 4.1 to 4.3.

Proposition 4.2. *(copied from Proposition 2.1) The plant output y_k of the system of Eq. 4.15 with the control law of Eq. 4.18 converges to a constant reference r if the closed-loop system matrix $\Phi^{rc}_{cl} = \Phi^{rc} + \Gamma^{rc}K^{rc}$ is stable, the matrix $\begin{bmatrix} \Phi^{rc} - I & \Gamma^{rc} \\ C_d & 0 \end{bmatrix}$ is invertible, and F^{rc} is defined by*

$$F^{rc} = \begin{bmatrix} K^{rc} & I \end{bmatrix} \begin{bmatrix} \Phi^{rc} - I & \Gamma^{rc} \\ C_d & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

with I and 0 denoting identity and zero matrices of appropriate dimensions, respectively.

Lemma 4.1. ***Closed-loop representation:*** *the discrete-time models from Eqs. 4.7 and 4.15 with control laws $u^{MC}_k = K^{MC}z^{MC}_k$ and $u^{rc}_k = K^{rc}z^{rc}_k$, respectively, have closed-loop representations given by:*

$$\begin{aligned} z^{MC}_{k+1} &= \left(\Phi^{MC} + \Gamma^{MC}K^{MC}\right) z^{MC}_k \\ z^{rc}_{k+1} &= \left(\Phi^{rc} + \Gamma^{rc}K^{rc}\right) z^{rc}_k. \end{aligned} \tag{4.30}$$

Lemma 4.2. ***Discrete-time Lyapunov stability of a plant:*** *A closed-loop representation given by $\Phi^{MC} + \Gamma^{MC}K^{MC}$ is globally asymptotically stable if and only if there exist matrices $P \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$ and $S \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$ that are positive definite (i.e., $P \succ 0$ and $S \succ 0$) that satisfy the following Discrete-Time Lyapunov Equation (DTLE) [107, Chapter 23]:*

$$P - (\Phi^{MC} + \Gamma^{MC}K^{MC})^T P \left(\Phi^{MC} + \Gamma^{MC}K^{MC}\right) \succ S.$$

Lemma 4.3. ***Stability under arbitrarily-switching plants:*** *Given a set of closed-loop representations of the form $\Phi^{MC} + \Gamma^{MC}K^{MC}$ and $\Phi^{rc} + \Gamma^{rc}K^{rc}$, $V_k = (z_k)^T P z_k$ is a CQLF if there exist positive-definite matrices $P \succ 0$, $S^{MC} \succ 0$, and $S^{rc} \succ 0$ such that P*

is the simultaneous solution of the DTLEs for all resource configurations:

$$\begin{aligned} P - (\Phi^{MC} + \Gamma^{MC} K^{MC})^T P \left(\Phi^{MC} + \Gamma^{MC} K^{MC}\right) &\succ S^{MC} \\ P - (\Phi^{rc} + \Gamma^{rc} K^{rc})^T P \left(\Phi^{rc} + \Gamma^{rc} K^{rc}\right) &\succ S^{rc}. \end{aligned} \tag{4.31}$$

The existence of a CQLF is a sufficient condition for stable arbitrary switching between the set of closed-loop representations [87].

Theorem 4.1. ***Common quadratic Lyapunov Function [17, 81]:*** *Consider the switching subsystems given by Eq. 4.30. If there exist positive-definite matrices $O \succ 0$, $S^{MC} \succ 0$, and $S^{rc} \succ 0$, and matrices Y^{MC} and Y^{rc} of appropriate dimensions such that the following LMIs holds:*

$$\begin{pmatrix} O & (\Phi^{MC} O + \Gamma^{MC} K^{MC} O)^T & O \\ (\Phi^{MC} O + \Gamma^{MC} K^{MC} O) & O & 0 \\ O & 0 & (S^{MC})^{-1} \end{pmatrix} \succ 0 \tag{4.32}$$

$$\begin{pmatrix} O & (\Phi^{rc} O + \Gamma^{rc} Y^{rc})^T & O \\ (\Phi^{rc} O + \Gamma^{rc} Y^{rc}) & O & 0 \\ O & 0 & (S^{rc})^{-1} \end{pmatrix} \succ 0, \tag{4.33}$$

then the closed-loop systems in Eq. 4.30 have a CQLF with a feedback gain $K^{rc} = Y^{rc} O^{-1}$. The existence of a CQLF guarantees the stability of the switched system under arbitrary switching.

Proof. Applying the Schur complement on Eq. 4.33 yields

$$O - (\Phi^{rc} O + \Gamma^{rc} Y^{rc})^T O^{-1} \left(\Phi^{rc} O + \Gamma^{rc} Y^{rc}\right) - O S^{rc} O \succ 0.$$

Defining $O = P^{-1}$ and $Y^{rc} = K^{rc} P^{-1}$ gives

$$P^{-1} - (\Phi^{rc} P^{-1} + \Gamma^{rc} K^{rc} P^{-1})^T P (\Phi^{rc} P^{-1} + \Gamma^{rc} K^{rc} P^{-1}) - P^{-1} S^{rc} P^{-1} \succ 0.$$

Pre- and post-multiplying by P gives

$$P P^{-1} P - P (\Phi^{rc} P^{-1} + \Gamma^{rc} K^{rc} P^{-1})^T P (\Phi^{rc} P^{-1} + \Gamma^{rc} K^{rc} P^{-1}) P - P P^{-1} S^{rc} P^{-1} P \succ 0,$$

which is equivalent to

$$P - (\Phi^{rc} + \Gamma^{rc} K^{rc})^T P \left(\Phi^{rc} + \Gamma^{rc} K^{rc}\right) \succ S^{rc}. \tag{4.34}$$

An identical procedure is applied to the LMI in Eq. 4.32:

$$P - (\Phi^{MC} + \Gamma^{MC} K^{MC})^T P \left(\Phi^{MC} + \Gamma^{MC} K^{MC}\right) \succ S^{MC}. \tag{4.35}$$

From Eqs. 4.34 and 4.35, it is clear that the systems defined by Eq. 4.30 have a CQLF defined in Lemma 4.3. This completes the proof. □

Remark 4.1. *The application of Lemma 4.3 in the systems described by Lemma 4.1 is conditioned to having identical augmented state vectors when random switching is triggered. This condition is guaranteed by the switching mechanism described in Section 4.5.1, specifically by Eqs. 4.24 and 4.29.*

Example 4.7. ***RC controller design:*** *Continuing with Example 4.2, Example 4.3, and Example 4.6, the two LMIs from Theorem 4.1 are simultaneously solved to design a controller for the RC. The resulting feedback gain is:*

$$K^{rc_1} = \begin{bmatrix} -43.08 & -9.75 & -3.46 & -2.99 \end{bmatrix}.$$

Applying Proposition 4.2, the feed-forward gain is given by:

$$F^{rc_1} = 49.79.$$

This section presented a control-design strategy for an RPC. This controller enhances QoC if the MC is used and provides stability if an RC is used. The next section provides an overview of the design flow of an RPC.

4.6 Summary of the design flow

The design of an RPC is summarized in the following steps. Given a sensing-to-actuating delay τ, a continuous-time plant as in Eq. 4.1, and γ^{ES} processing resources running sa sporadic applications, an RPC with a maximum of γ^{MC} sensing units is designed following these design steps:

1. **Modelling MC**. Find the sampling period h^{MC} using Eq. 4.3. Discretize the continuous-time model with Eq. 4.5. Construct the discrete-time model using Eqs. 4.6-4.8.

2. **Modelling RC**. Based on the sa sporadic applications, decide how many RCs are required; this is a design parameter. For each RC, determine the available processing resources γ^{rc}; find the sampling period h^{rc} with Eq. 4.10; discretize the continuous-time model with Eq. 4.5. Construct the discrete-time models using Eqs. 4.14-4.16.

3. **Controller design MC**. Find a feedback gain K^{MC} such that the QoC is optimized. To do so, we use the PSO algorithm presented in Section 2.5. Find the feed-forward gain F^{MC} using Proposition 4.1.

4. **Controller design RC**. Find the set of feedback gains K^{rc} solving the LMIs stated in Theorem 4.1 for all RCs. Design a feed-forward gain F^{rc} for each RC using Proposition 4.2.

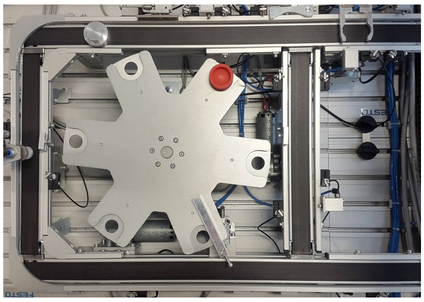
Following these design steps, an RPC is designed. Such a controller is capable of stable switching between an MC and one or more RCs. In the next sections, a complete example of an RPC implementation is shown.

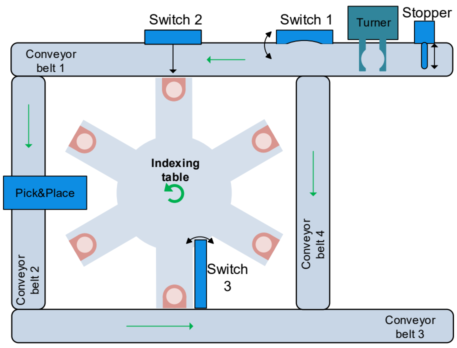
4.7 Case study: xCPS

We evaluate our approach through simulations based on the experimental platform xCPS described in Section 2.2 [3, 2]. An Image-Based Control (IBC) and multiple sporadic applications share a multi-core embedded system in xCPS. The applications are sporadically triggered allowing their cores to be temporarily reused by the controller in order to enhance QoC. The xCPS platform is an industrial assembly line simulator which is used for teaching and research purposes. The machine assembles or disassembles circular complementary pieces that come in two shapes: lower and upper parts. The assembly section of xCPS (Fig. 4.6a) is considered in this case study. The following subsections describe the xCPS assembly process.

4.7.1 Assembly applications and resources

A schematic of the assembly hardware is depicted in Fig. 4.6b. Four sporadic tasks are involved in the assembly process: a Supervisory Control Application (SCA), a Turner Application (TA), an Indexing Table Application (ITA), and an IBC. The assembly process is controlled by the SCA, which is activated whenever a new assembly piece enters *conveyor belt 1*. The SCA also generates set-points for the controllers, activates the system actuators (such as *switches*, the *stopper* and the *pick and place unit*), and distributes the available resources among the assembly applications, i.e., it regulates the activation policies of the sensing cores. The TA corrects the orientation of the assembly pieces in case it is necessary, by manipulating the *turner* actuator. The ITA consists of a local controller that aligns an *indexing table* either with *switch 2* or with a *pick and place* actuator. The IBC regulates the speeds of the assembly pieces on the conveyor belts using DC motors. A camera and an image-processing algorithm are used as sensor to measure the speed of the assembly pieces on the belts. The speed is kept low when a piece is going through the actuators (e.g., turner, switch 2, and pick and place) and high otherwise. Therefore, the highest controller performance is needed when the speed of the conveyor belt is to be reduced or increased. The SCA is triggered


(a) Camera view.



(b) Schematic.

Figure 4.6: Assembly section of xCPS.

sporadically based on the arrival of a new piece. The activation time of ITA and TA is defined by the SCA. The controller always remains active.

A five core multiprocessor system ($\gamma^{ES} = 5$) is considered to simulate the assembly applications. The three aforementioned sporadic applications ($sa = 3$) and the controller are executed on these five cores. Each application runs on a separate core ($\gamma^{sa} = 3$) in case they need to be executed at the same time.

4.7.2 Assembly process

The assembly process begins when a piece is fed onto *conveyor belt 1* and it is blocked by a *stopper*. The SCA is immediately triggered to generate controller references, activation policies of the cores, and activation times of the actuators that guarantee the correct assembly of the piece. The SCA retracts the *stopper* and the piece moves through a *turner* activating the TA. Lower pieces go to an *indexing table* using *switch 2* and trigger the ITA, whereas upper pieces go to *conveyor belt 2*. A *pick and place* actuator grabs the upper pieces from *conveyor belt 2* and pushes them onto the lower parts on the *indexing table*. Assembled pieces are moved by *switch 3* onto *conveyor belt 3*. The system also rejects assembly pieces using *conveyor belt 4* and *switch 1*.

Since the conveyor belt speeds are regulated by the IBC, the time elapsed between a piece leaving the *stopper* and reaching the *turner* or the *indexing table* is known. Therefore, the activation time of the ITA and TA can be predicted. This implies that the switching initiation time is known in advance, which allows to properly initialize the switching. However, the SCA is triggered by the arrival of a new piece; therefore its activation time cannot be predicted. This is particularly important for the design of the RPC in the next section. The throughput of the machine is directly affected by the settling times of the controller: a shorter settling times means that the blocks reach each stage of the assembly process faster, which increases the number of assembled blocks per time unit. Therefore, optimizing the QoC potentially increases the system throughput [13].

4.8 Simulation results

In this section, four different sensing configurations of the IBC are studied. Their QoC is then evaluated using the model of xCPS. The settling time of the controller is used as QoC metric since it has a potential impact on xCPS throughput. Note that although the example of Section 2.2 is used as motivation, a different set of parameters is used in the setup and results below.

4.8.1 IBC design

The IBC manipulates the speed of the assembly pieces on the conveyor belts by adjusting the input voltage to the DC motors. The parameters of the motor model are:

$$A_c = -49.50, B_c = 0.01, C_c = 1.$$

The reference r is the desired speed of the DC motor. The controller is designed to reach such a reference as fast as possible (i.e., the shortest settling time) using the algorithm from Section 2.5. The sensing-to-actuating delay is $\tau = 0.125\ s$. The system shows a rise-time of $R_t = 44\ ms$. In absence of delay, the sampling period could be defined with the rule of thumb of Eq. 2.3, which results in $h_{Rt} = 4.4\ ms$. However, due to the sensing-to-actuating delay, the shortest sampling period that can be obtained with a single processing resource for the controller (serial configuration) is given by $h_s = 0.125\ s$. Since $h_s >> h_{Rt}$, a pipelined controller can improve the control performance.

Given the sporadic applications (SCA, TA, ITA with $sa = 3$) and the multi-core platform ($\gamma^{ES} = 5$), multiple IBCs are designed with different processing resources: a serial configuration, a pipelined configuration, a four-core RPC, and a five-core RPC. The serial configuration is used as a comparison benchmark; the pipelined configuration is designed to show the benefits of applying the method of Chapter 2; the four-core RPC is designed to show the benefits of dynamically allocating the processing resources of the tasks whose activation time can be predicted to the RPC; the five-core RPC is designed to show the benefits and risks of dynamically allocating the processing resource of a sporadic task whose activation time cannot be predicted (i.e., SCA) to the RPC. In the following paragraphs, we explain each of the resource configurations designed.

4.8.1.1 Serial configuration SC

A serial configuration (denoted by SC) uses one core for executing the controller as shown in Fig. 4.7a. The sensing-to-actuating delay $\tau = 0.125\ s$ implies a sampling period $h^{SC} = \tau = 0.125\ s$. Since the sampling period is known, Eq. 4.5 is used to discretize the continuous-time model. The augmented state vectors are redefined using Eqs. 4.6 and 4.8 to include the delay in the state-space notation. An IBC with minimum settling time is designed by using the PSO algorithm from Section 2.5. The resulting feedback gain is:

$$K^{SC} = \begin{bmatrix} -0.22 & -210 \end{bmatrix} \times 10^{-4}.$$

Proposition 4.1 is used to find the feed-forward gain $F^{SC} = 0.51$.

4.8.1.2 Pipelined configuration PC

Using the two statically available processing resources in the embedded system, a pipelined controller with $\gamma = 2$ (denoted PC) is designed, as illustrated in Fig. 4.7b. Since the number of available processing resources for sensing is constant, the controller is designed using the guidelines of Chapter 2, considering that $\gamma = 2$. The resulting sampling period is $h = 62.5 \times 10^{-3}\ s$. As for the SC, the PSO algorithm from Section 2.5 is used to find the feedback gain with minimum settling time:

$$K^{PC} = \begin{bmatrix} 0.23 & 9.94 & -78.76 \end{bmatrix} \times 10^{-3}.$$

The feed-forward gain is given by $F^{PC} = 0.53$.

4.8.1.3 Four-core RPC (RPC_4)

Fig. 4.7b shows that the ITA and TA are applications with a low frequency of appearance. Such applications are triggered by the SCA, making it possible to schedule them in such a way that their processing resources are temporarily reused by an RPC. This scenario is shown in Fig. 4.7c. For this example, the reconfiguration mechanism is constantly used such that the MC is only used when the best control performance is required i.e., when the reference speed is changed. For the rest of the operation, an RC is used. This reduces the usage of processing resources of the RPC, which makes further room for the execution of the other applications in the system. For example, in Fig. 4.7c processing resource three is temporarily activated to complete an actuation operation at time 281.3 ms. This results in the actuation period of the MC being active in the range $[250, 312.5)$ ms. For the next actuation, the controller switches back to an RC with two resources. As a result, the fourth resource in the MC is never activated. The figure shows several other of such short activations of the MC, leading to temporary activations of the third core, but never to the activation of the fourth core.

The procedure presented in Section 4.6 is used to design an RPC denoted by RPC_4. RPC_4 is composed of two sensing configurations: an MC with four processing resources ($\gamma^{MC} = 4$) and an RC rc_1 with two processing resources ($\gamma^{rc_1} = 2$). The MC has a sampling period $h^{MC} = 31.25 \times 10^{-3}$ s. Using the same procedure described for the previous controllers, the feedback gain with minimum settling time is given by:

$$K^{MC} = \begin{bmatrix} -0.33 & -2.48 & -18.45 & -47.89 & -226.07 \end{bmatrix} \times 10^{-3}.$$

The feed-forward gain is $F^{MC} = 0.64$.

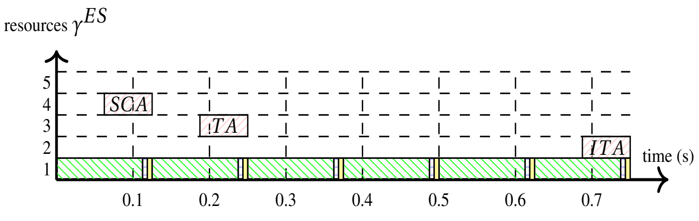
The sampling period of the RC is $h^{rc_1} = 62.5 \times 10^{-3}$ s. The LMIs from Theorem 4.1 are solved with the optimization toolbox Yalmip [83] and a semi-definite quadratic solver SDPT3 [121]. The resulting feedback gain is

$$K^{rc_1} = \begin{bmatrix} 0.52 & 3.89 & 28.18 & 68.49 & 336.07 \end{bmatrix} \times 10^{-3}.$$

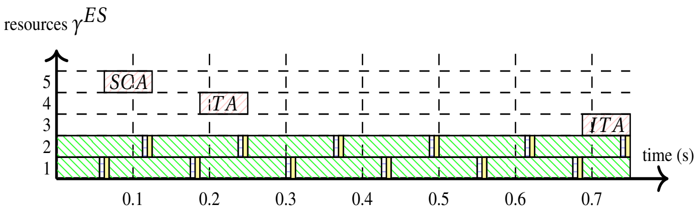
The feed-forward gain is $F^{rc_1} = 0.28$. The camera has to support an acquisition time of 31.25 $ms \leq h_{ac} \leq 62.5$ ms and an external trigger.

4.8.1.4 Five-core RPC (RPC_5)

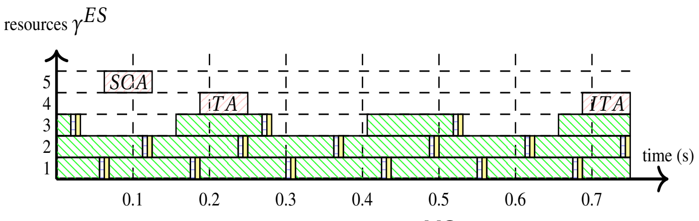
In order to reuse all resources of the sporadic applications, the procedure from Section 4.6 is used for designing an RPC (denoted by RPC_5) that uses up to five processing resources. Since the activation time of the SCA is not predictable, two cases are considered: when no interruption occurs in the sensing pipeline (e.g., SCA triggers



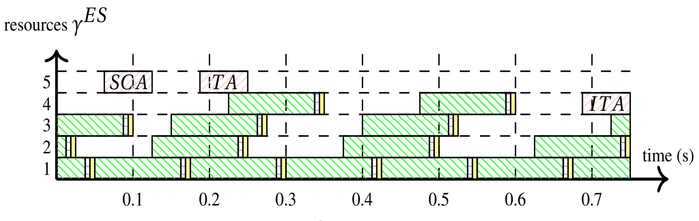
(a) Serial configuration $\gamma = 1$.



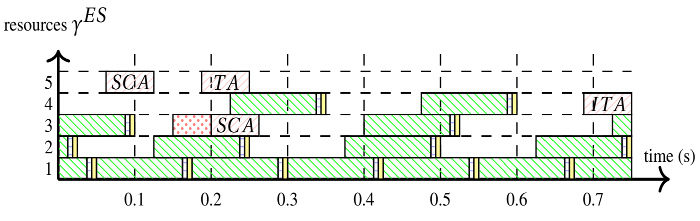
(b) Pipelined configuration with $\gamma = 2$.



(c) RPC_4 with $\gamma^{MC} = 4$.



(d) RPC_5 with $\gamma^{MC} = 5$, no interruption from SCA.



(e) RPC_5 with $\gamma^{MC} = 5$ with interruption from SCA. No sensing information is delivered at time 0.275 s (see red dotted square).

Figure 4.7: Resource allocation in xCPS when a new lower-part piece arrives to *conveyor belt 1*. It is assumed that all the controllers were already running when xCPS starts.

when its sensing core is available as shown in Fig. 4.7d) and when interruptions do occur (e.g., SCA preempts the sensing operation as shown in Fig. 4.7e). Like in the previous case, the MC is only used when the control performance needs to be enhanced (when the reference changes) while an RC is used otherwise. For example in Fig. 4.7d, processing resources are temporarily scheduled such that the MC completes actuation operations at times 250, 275, and 300 ms. For the next completion of an actuation operation (at time 350 ms) an RC is then used again. As for RPC_4, this results in core five never being active for the RPC.

RPC_5 is composed of three sensing configurations: an MC with five pipes ($\gamma^{MC} = 5$), and two RCs: rc_1 and rc_2. rc_1 uses two pipes ($\gamma^{rc_1} = 2$), while rc_2 uses three pipes ($\gamma^{rc_2} = 3$). For the MC, the parameters are $h^{MC} = 25 \times 10^{-3} s$, $F^{MC} = 0.71$, and

$$K^{MC} = \begin{bmatrix} 0.28 & 1.40 & -7.02 & -17.46 & -99.57 & -314.11 \end{bmatrix} \times 10^{-3}.$$

For rc_1, the parameters are $h^{rc_1} = 62.5 \times 10^{-3} s$, $F^{rc_1} = 0.34$, and

$$K^{rc_1} = \begin{bmatrix} 0.11 & 0.05 & 4.26 & 12.17 & 67.13 & 22.85 \end{bmatrix} \times 10^{-3}.$$

For rc_2, the parameters are $h^{rc_2} = 50 \times 10^{-3}\ s$, $F^{rc_2} = 0.34$, and

$$K^{rc_2} = \begin{bmatrix} -0.01 & 0.04 & 4.39 & 12.38 & 68.66 & 227.11 \end{bmatrix} \times 10^{-3}.$$

The camera has to support an acquisition time of 25 $ms \leq h_{ac} \leq 62.5\ ms$ and an external trigger.

4.8.2 Controller simulations

The controllers designed in the previous subsection are simulated when a new lower-part piece arrives to *conveyor belt 1*. For simplicity, only the speed of the first conveyor belt is illustrated in the figures. The reference is initially kept low while the piece passes through the *turner*. When the TA is finished, the speed is temporarily increased to reach the next stage of the assembly process. The piece then moves to the *indexing table* activating the ITA. The distribution of applications on the embedded system is shown in Fig. 4.7 for the different sensing configurations. The activation policy of the cores guarantees that the MC is active whenever the reference speed is changed, so that the block reaches the desired position faster. In absence of disturbances, sensing errors, and modelling errors, zero steady-state error is achieved in these cases.

4.8.2.1 *SC* vs *PC*

The model output and the controller input are plotted in Fig. 4.8. It is clear that the controller with pipelined sensing (red dashed line) outperforms the serial controller (purple dash-dotted line). The controller in SC actuates once every sensing-to-actuating delay τ, whereas the controller in PC actuates twice in the same period. As a consequence, a more aggressive controller input is achieved by the PC controller. At time 0.25 s, the controller input of the PC controller is initially higher than the SC controller but after some samples they both converge to the same value. The PC controller achieves a settling time 30% shorter than the one achieved by the SC controller. Fig. 4.8 also shows that a change in the controller input is not immediately seen in the motor speed because of the delay introduced by the sensing operation.

Table 4.2: QoC comparison of sensing configurations.

Sensing configuration	settling time ms	settling time enhancement %
SC	78	0
PC	54	30
RPC_4 with $\gamma^{MC} = 4$	30	44
RPC_5 with $\gamma^{MC} = 5$	24	20

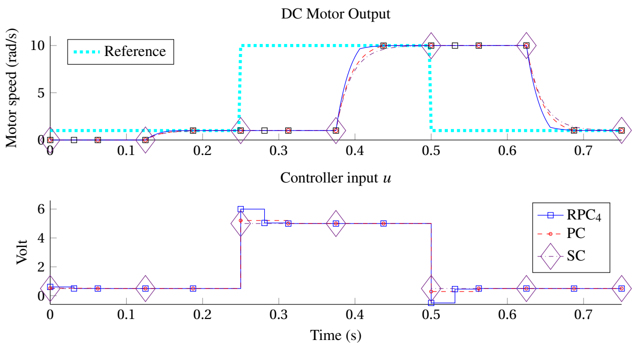


Figure 4.8: Comparison: SC, PC, and RPC with $\gamma^{MC} = 4$ (RPC_4). Markers denote the actuation instants.

4.8.2.2 PC vs four-core RPC

The model output and controller input are plotted in Fig. 4.8. The QoC of the RPC is enhanced using MC when the plant is away from the reference. The RC is used when the plant is on the reference. As a result, RPC actuates more frequently when the reference changes. The RPC controller outperforms the PC controller by reaching the reference 44% faster in all set-point changes. This result is summarized in Table 4.2.

4.8.2.3 Four core RPC vs five core RPC

If the arrival time of a new block is known in advance, the SCA can be scheduled in such a way that the RPC with $\gamma^{MC} = 5$ uses the MC when there is a deviation from the reference and the RCs are used when the plant is on the reference. The simulation results in Fig. 4.9 show such a scenario. RPC_5 with $\gamma^{MC} = 5$ (red dashed line) reaches the reference 20% faster than RPC_4 (blue continuous line). This result is summarized in Table 4.2 again. However, if an unexpected block enters the plant, the SCA is triggered potentially causing sensing information to be dropped, an actuation operation to be omitted, and a forced switch to a resource configuration with fewer cores. Additionally, the omitted actuation forces the controller to use the previously applied actuation for longer than designed. This can cause intermediate actuation periods to appear in the controller, which might have a negative affect on the control stability. In Fig. 4.9, RPC_5 (purple dash-dotted line) is forced to switch from the MC to an RC processing resources because of an unexpected activation of SCA at time 0.2 s. Since an RC was activated, no intermediate sampling periods appeared. However, the QoC is severely affected.

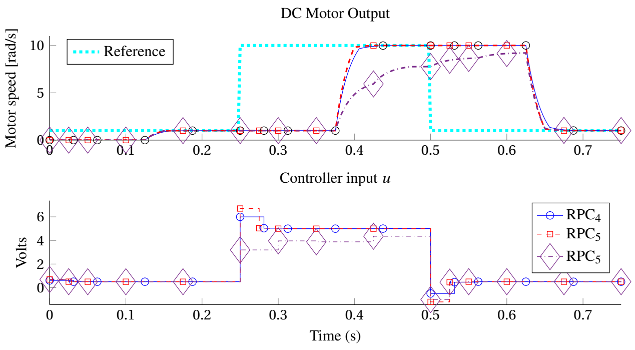


Figure 4.9: Comparison between RPC_4 and RPC_5 (red dashed line). RPC_5 also is simulated when an RC is used while the QoC needs be to enhanced (purple dash-dotted line).

4.8.3 Discussion

From the simulations described in this section, the following insights are deduced. First, pipelined control outperforms serial control due to the higher sampling rate, at the cost of more resources used. Second, an RPC enhances QoC if the MC is active while reference changes occur. If another resource configuration is active, the controller is still stable with a lower QoC. Third, an RPC improves QoC if the arrival times of the sporadic applications are known, which allows to properly initialize the switching mechanism. In case one or multiple sporadic applications are not predictable, it is recommended to not include such processing resources in the RPC because it might result in QoC deterioration rather than improvement, or even control instability.

4.9 Summary

This chapter presented reconfigurable pipelined control for DISC. A pipelined controller reduces the sampling period of a DISC using additional sensing resources in a pipelined fashion, while a reconfigurable pipelined controller reduces the sampling period further by temporarily reusing the resources from other applications running on the same platform. We have introduced a state-based modelling strategy and a controller-design strategy for reconfigurable pipelined control. Simulation results show that a reconfigurable pipelined controller has better QoC than a static pipelined controller. This is valid if the maximal configuration is used when the QoC needs to be enhanced, more resources are available to the RPC than to the pipelined controller, and the reconfiguration mechanism is timely initialized. Otherwise, an RPC is still stable, at the cost of certain QoC degradation.

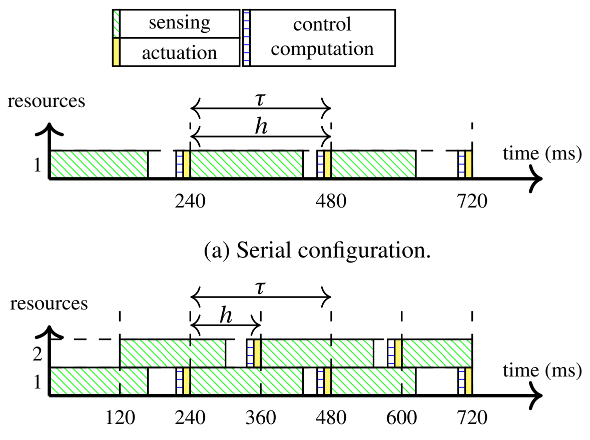
The method presented in this chapter uses the worst-case execution time of the sensing algorithm to design pipelined-sensing controllers. In the next chapter, we consider variable sensing delays in the design of pipelined controllers, which can be used to further improve QoC.

Chapter 5

Implementation-aware variable-delay pipelined control

5.1 Problem formulation 110
5.2 Related work 112
5.3 Variable-delay predictor controller 113
5.4 Variable-delay control design 120
5.5 Case study: xCPS 125
5.6 Simulation results 127
5.7 Summary 131

The design strategies of DISC (commonly) consider that the signal-processing algorithm used for obtaining the sensing information has a constant delay. However, in practice, the sensing delay of an algorithm varies depending on the state of the platform (e.g., cache misses, data availability, scheduling policies) and the data being processed (e.g., varying number of regions of interest, noise). Therefore, these control-design strategies consider the constant worst-case execution time of the processing algorithm (see Fig. 5.1). Such a worst-case execution time is unlikely to happen and considerably larger than the most common delays, which leaves room for further improvement of QoC. Moreover, modern embedded platforms allow to characterize the sensing delay at design time obtaining a delay histogram, and at run-time measuring its precise value. In this chapter, we exploit this knowledge to design variable-delay pipelined



(a) Serial configuration.

(b) Pipelined configuration with two sensing cores.

Figure 5.1: Examples of resource configurations of DISC. The worst-case execution time delay τ of the signal-processing algorithm keeps the sampling period h constant.

controllers. Unlike the previous chapters, the model of the variable-delay pipelined control is not based on the worst-case execution time of the processing algorithm. Therefore, a different modelling strategy is used in this chapter. Additionally, since the control performance strongly depends on the model quality, we present a simulation benchmark that uses the model uncertainties and the delay histogram to obtain bounds on control performance. Our benchmark is used to select a variable-delay controller and a resource configuration that outperform a constant worst-case-delay controller.

The contents of this chapter were published in [88].

5.1 Problem formulation

The design of a variable-delay controller requires extended knowledge about the variability of the delay (e.g., worst case, best case, modes) and a control strategy capable of coping with such a variation.

The variability of the sensing algorithm can be captured with an implementation-dependent histogram of delays. An example of such a histogram is shown in Fig. 5.2. The sensing delay was referred to with τ_s in Chapter 2, but for simplicity of notation, we omit the subscript s in this chapter. The delay shows an upper bound τ (worst case),

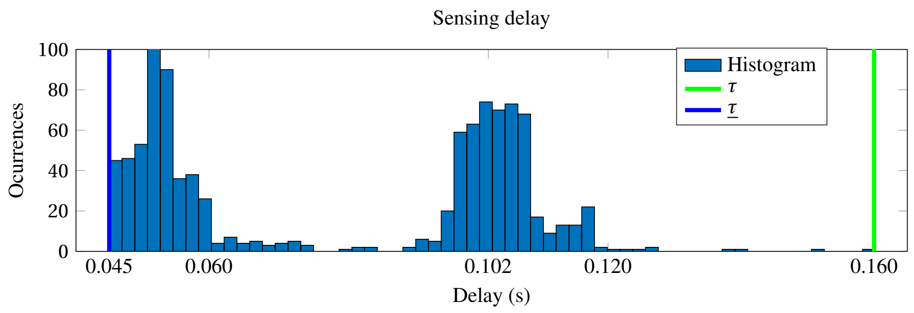


Figure 5.2: Example histogram of variable sensing delay in an experiment with 10000 executions of an image-processing algorithm.

a lower bound $\underline{\tau}$ (best case), and one or more most-likely cases (modes) [45]. A DISC is normally designed using the upper bound of the delay to guarantee a constant sensing delay for the controller (worst-case design [75, 144]). However, as shown in Fig. 5.2, the worst-case execution may be unlikely to happen, which makes a design based on this delay conservative.

Control-design strategies for variable-delay controllers have been widely studied in the literature (e.g., [39, 40, 46]). These approaches are mainly focused on analysing the robustness of a controller under the assumptions that the delay varies in a bounded range (i.e., there is a worst-case and a best-case delay) and it cannot be measured on-line (i.e., the delay value is not available before the controller is computed, e.g., because of a transfer of sensed data over a wireless network). Although these are valid assumptions in many applications, in DISC, the delay can be commonly characterized off-line with an implementation-dependent delay histogram. This information can be used at design time to create an implementation-dependent controller. Additionally, once the signal-processing algorithm has finished its execution, existing timing mechanisms (e.g., built-in timers) can be used to on-line measure the execution-time of the algorithm at micro-second-level accuracy (e.g., [10]). This information can be made available to the controller at run-time to compensate for the variability of the delay, which may improve its QoC.

Our contribution is an implementation-aware variable-delay pipelined control design. The novelty of the design lies in the combination of implementation-dependent information (i.e., histogram of delays, measured delay, and resource configurations) with model-based predictions to compensate for the variable delay while improving QoC. We also provide a simulation benchmark that uses the model uncertainties and the delay histogram to select a controller and a resource configuration that improves QoC.

5.2 Related work

Designing implementation-aware variable-delay controllers requires knowledge about the histogram of the delay values. Strategies for obtaining such histograms have been reported in [53, 4] for single-mode histograms, and in [45] for multi-mode histograms. Single-mode histograms consider that the variations in sensing delay of a target application result from variation in platform aspects such as scheduling policies and cache misses. No explicit data-dependent loops are included in the tests. Multi-mode histograms are obtained in multimedia applications, where the decoding tasks have a workload that is significantly affected by the data being decoded. The delay of signal-processing algorithms may also be influenced by the data being processed (e.g., possible changes in the number of objects being detected). Therefore, we consider delay histograms with one or multiple peaks.

Variable-delay control has been widely studied in the discrete-time domain using robust static feedback control [39, 40, 51], robust predictor-based control [85, 41, 46], and event-driven control approaches [82, 96, 125]. Robustness approaches provide stability guarantees for a controller given a variable range of delay variation. Event-driven control computes the controller input as soon as the data arrives to the controller. These approaches are motivated by networked control, where a network introduces sensing and actuation delay into the control loop. Although the sensing delay may be measured before the control action is computed, the actuation delay is unavailable at the moment of computing the control action and/or the clocks of devices in a network may not be synchronized. Therefore, these strategies assume that the delay is unmeasurable but bounded to a range. Additionally, these approaches provide QoC improvements in terms of quadratic cost or asymptotic stability, which are not intuitively related to real-time metrics such as settling time, overshoot or rise time. While these strategies are applicable to variable-delay DISC, they provide conservative design solutions in terms of QoC as we show later. Exploiting the on-line measurability of the delay on today's multiprocessing embedded systems can be used to improve QoC in terms of real-time metrics such as settling time.

Strategies to compensate for a measurable variable delay using model-based predictions use Smith Predictors [130], Model Predictive Control (MPC) [113], and Smith Predictors with MPC [63]. The Smith-Predictor scheme compensates for the variable delay using the system model and on-line measurements of the delay. Therefore, this strategy may be suitable for the pipelined control case. MPC solves an optimization problem every sampling period. MPC may be applied to data-intensive control, taking into account the additional delay required to solve the optimization algorithm on-line. An implementation-aware variable-delay controller was presented in [42]. The authors use a single-mode probability distribution of the sensing delay to compute the probability of a deadline miss. Real-time QoC metrics are used to quantify the QoC with deadline misses. However, no on-line delay measurements are used to compensate for the delay. To the best of our knowledge, no other implementation-aware approaches for measurable delays were reported in the literature at the time of first publishing this work.

We propose an alternative to the existing variable-delay control-design strategies. Our approach designs a set of controllers based on implementation-dependent histograms of delays, and on-line measurements of the delay. To do so, we consider the delay-free case of the system (i.e., without sensing delay) to design controllers with actuation periods smaller than the worst-case execution time of the sensing delay. We then compensate for the variable delay with a modification of the predictor approach proposed in [46]. This predictor estimates the state vector using model-based predictions with a constant delay. The proposed predictor in this work estimates the state vector using a variable-delay estimator. Ideally, our estimator cancels the effect of the delay resulting in a QoC similar to the delay-free case. However, in practice prediction errors occur due to model uncertainties. Therefore, we introduce a simulation benchmark that obtains QoC bounds based on model uncertainties. The benchmark is used to select a resource configuration and one controller such that the QoC is improved compared to a constant worst-case design.

5.3 Variable-delay predictor controller

A variable-delay predictor controller has the structure presented in Fig. 5.3. The sensing operation delay shows a time-varying behaviour with one or several modes, as the example in Fig. 5.2. The variable delay is compensated by the prediction block, which estimates the state vector to create a delay-free estimation. This cancels the effect of the variable delay from the control perspective. The controller uses this estimation to compute a new control input. Since no delay is seen from the controller perspective, the controller sampling period is no longer limited by the worst-case execution time of the sensing delay (in contrast to the previous chapters). We subdivide the explanation of the controller in six subsections: run-time behaviour, modelling, control block, predictor block, prediction length, and prediction quality.

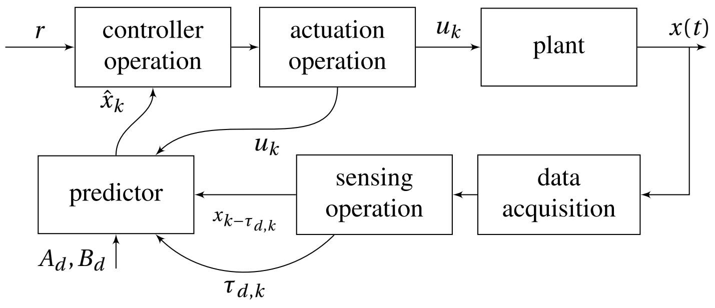
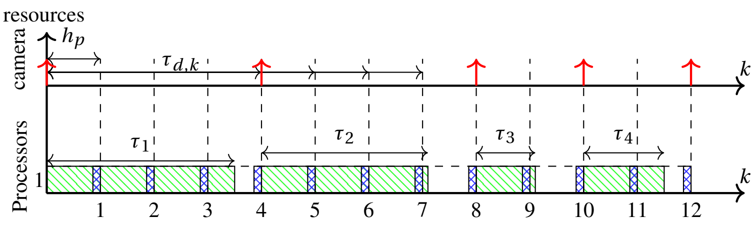


Figure 5.3: Variable-delay predictor controller.

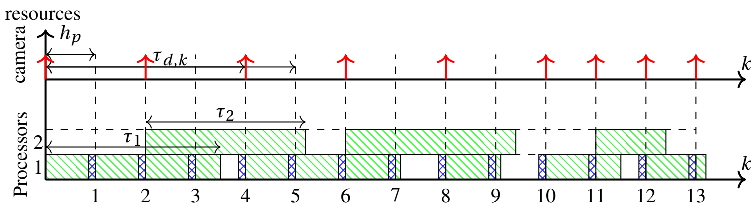
5.3.1 Run-time behaviour

An example of the resource usage of a predictor controller is shown in Fig. 5.4a for a serial case and in Fig. 5.4b for a pipelined case with two processing units. The delays of consecutive sensing operations are denoted by τ_1, τ_2, etc. The controller is composed of a sensing operation (i.e., a signal-processing algorithm), a prediction operation (see Proposition 5.1), a control-computation operation (i.e., Eq. 5.13), and an actuation operation. The actuation period $h_p \in \mathbb{R}^+$ denotes the interval between two consecutive controller updates by the data-intensive controller. Note that an actuation period is used in this chapter (instead of a sampling period) because the sensing operation can start at irregular time intervals. The data acquisition, prediction, control computation, and actuation are periodic operations with period h_p. This timing is assumed strictly constant. Note that not all data acquired by the data-intensive controller is used by the sensing operations due to the unavailability of processing resources. In case the sensing operation is still running when the periodic operations need to start (e.g., near the end of time instant $k = 1$ in the example of Fig. 5.4), the sensing operation is temporarily pre-empted. The duration of such pre-emptions is assumed to be included in the sensing delay τ_i. The execution times of the sensing operations τ_i fall within the bounds of the sensing delay $[\underline{\tau}, \tau]$. A new sensing operation on any of the allocated processing resources starts its execution when the previous sensing operation on that resource is finished, at the next multiple of h_p (i.e., kh_p for some natural number k). The sensing operation is therefore non-periodic.

The use of pipelined sensing control creates communication overhead when sharing information between processing resources. For example, in Fig. 5.4b, the upper pipe sends the output of the image-processing algorithm to the lower pipe that computes the periodic operations. Typically the size of this information is limited to a few floating-point numbers (e.g., one floating-point number per measured output in our case study).



(a) Serial variable-delay control.



(b) Pipelined variable-delay control with two sensing cores.

Figure 5.4: Variable-delay control. Every h_p seconds, data is acquired by the data-intensive control. Each vertical red arrow represents a data frame that is being processed by the sensing operation in the controller. Note that some acquired data is not used by the sensing operations. Blue blocks with a crossed pattern represent the prediction, control computation, and actuation operations. The sensing delay varies with the sequence $\tau_1, \tau_2, \tau_3, \tau_4, \ldots$. The prediction length $\tau_{d,k}$ denotes the age (in samples) of the data acquired by τ_i. It is therefore continuously updated from the completion of τ_1 (i.e., $\tau_{d,4} = 4$) until the completion of τ_2 (i.e., $\tau_{d,6} = 4$ in the pipelined case while $\tau_{d,8} = 4$ in the serial case).

Therefore, this overhead is commonly negligible compared to the image-processing delay. Additionally, note that at time $k = 10$ in the pipelined case of Fig. 5.4b, both processors finish their computation in the same time slot. The prediction then uses the output based on the most recent image, which in this case comes form the lower processor.

5.3.2 Plant modelling

We consider the following continuous-time system:

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t), \end{aligned} \tag{5.1}$$

with $x(t) \in \mathbb{R}^{n\times 1}$ the state vector, $u(t) \in \mathbb{R}$ the control input, and $y(t) \in \mathbb{R}$ the plant output, $A_c \in \mathbb{R}^{n\times n}$, $B_c \in \mathbb{R}^{n\times 1}$, $C_c \in \mathbb{R}^{1\times n}$ the state, input, and output matrices, respectively, and $t \in \mathbb{R}^{\geq}$ the time. $n \in \mathbb{Z}^{+}$ is the number of states. We also consider that A_c is a Hurwitz matrix (i.e., a stable matrix). The discrete-time version of Eq. 5.1 is given by:

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_k \\ y_k &= C_c x_k, \end{aligned} \tag{5.2}$$

where $x_k \in \mathbb{R}^{n\times 1}$ and $y_k \in \mathbb{R}$ are the discrete-time state and output vectors, respectively, $x_k := x(kh_p)$ and $y_k := y(kh_p)$ with $k \in \mathbb{Z}^{\geq}$. $u_k \in \mathbb{R}$ is the control input which is implemented with a ZOH resulting in $u(t) := u(kh_p)$ for $t \in [kh_p, (k+1)h_p)$. $A_d \in \mathbb{R}^{n\times n}$ and $B_d \in \mathbb{R}^{n\times 1}$ are the discrete-time state and input matrices, further defined by [104, Chapter 2]:

$$A_d = e^{A_c h_p}, B_d = \int_0^{h_p} e^{A_c s} B_c ds. \tag{5.3}$$

$h_p \in \mathbb{R}^{+}$ is the actuation period of the controller, which is a design parameter chosen as described in Section 5.4.2.

5.3.3 Controller block

Following the arguments of Section 1.2, the QoC in our application is defined as

$$QoC = S_t^{-1},$$

where $S_t \in \mathbb{R}^{+}$ is the controller settling time. The controller is then designed to maximize the QoC. Note that, in the proposed design, the current state vector x_k is not available due the sensing delay. The controller is designed based on the assumption that the predictor block produces a predicted state $\hat{x}_k \in \mathbb{R}^{n\times 1}$, which is considered equal to the current state vector. From the control perspective the sensing delay is therefore cancelled by the predictor. The controller (see Fig. 5.3) is then designed for the plant of Eq. 5.2 without taking into account the effect of the sensing delay.

In this chapter, we consider control robustness as the capacity of the controller to reach a desired reference within a 2% margin in the presence of model uncertainties. Control robustness is later considered during the design of the variable-delay controller.

5.3.4 Predictor block

The model of Eq. 5.2 is used by the predictor block (see Fig. 5.3) to estimate the state vector at time k. The predictor operation uses Proposition 5.1 below to estimate the state vector. The predictions are based on the available information: a prediction length $\tau_{d,k} \in \mathbb{Z}^+$, the latest state-vector measurement $x_{k-\tau_{d,k}}$, past controller inputs $u_{k-\tau_{d,k}+j}$ (with $0 < j \leq \tau_{d,k}$), and the discrete-time model of Eq. 5.2. The prediction length describes in samples the age of the measured data used for the k^{th} control operation. $\tau_{d,k}$ is explained in detail in Section 5.3.5. Our predictor is inspired by the one presented in [46], which considers $\tau_{d,k}$ constant (and not measurable) in the predictions. A robustness-analysis technique is then applied to the controller to guarantee its stability provided the bounds on variation of the delay $\underline{\tau}$ and τ. Our predictor uses on-line measurements of the delay to compensate for the variable delay, which creates a prediction with a variable number of actuation periods (i.e., in Proposition 5.1 below, $\tau_{d,k}$ is updated every time a prediction is needed). This (ideally) cancels the effect of the sensing delay, guaranteeing the stability of the controller.

Proposition 5.1. *Predictor with variable prediction length: Given the discrete-time plant of Eq. 5.2 with sensing data measured $\tau_{d,k}$ actuation periods ago, the state vector at time k are estimated using:*

$$\hat{x}_k = (A_d)^{\tau_{d,k}} x_{k-\tau_{d,k}} + \sum_{j=0}^{\tau_{d,k}-1} (A_d)^{\tau_{d,k}-1-j} B_d u_{k-\tau_{d,k}+j}.$$

Proof. The state vector of Eq. 5.2 is shifted into past states, until a dependence with the measured state vector $x_{k-\tau_{d,k}}$ is found:

$$\hat{x}_k = A_d \hat{x}_{k-1} + B_d u_{k-1}$$

$$\hat{x}_{k-1} = A_d \hat{x}_{k-2} + B_d u_{k-2}$$

$$\vdots$$

$$\hat{x}_{k-\tau_{d,k}+2} = A_d \hat{x}_{k-\tau_{d,k}+1} + B_d u_{k-\tau_{d,k}+1} \tag{5.4}$$

$$\hat{x}_{k-\tau_{d,k}+1} = A_d x_{k-\tau_{d,k}} + B_d u_{k-\tau_{d,k}}. \tag{5.5}$$

A backward substitution, i.e., Eq. 5.5 into Eq. 5.4, yields:

$$\hat{x}_{k-\tau_{d,k}+2} = A_d \left(A_d x_{k-\tau_{d,k}} + B_d u_{k-\tau_{d,k}} \right) + B_d u_{k-\tau_{d,k}+1},$$

which is equivalent to

$$\hat{x}_{k-\tau_{d,k}+2} = (A_d)^2 x_{k-\tau_{d,k}} + A_d B_d u_{k-\tau_{d,k}} + B_d u_{k-\tau_{d,k}+1}.$$

This backward substitution strategy is repeated until $\tau_{d,k}$ steps are achieved:

$$\begin{aligned}\hat{x}_{k-\tau_{d,k}+\tau_{d,k}} &= (A_d)^{\tau_{d,k}} x_{k-\tau_{d,k}} + (A_d)^{\tau_{d,k}-1} B_d u_{k-\tau_{d,k}} \\ &+ (A_d)^{\tau_{d,k}-2} B_d u_{k-\tau_{d,k}+1} + \cdots + (A_d)^0 B_d u_{k-1}.\end{aligned} \tag{5.6}$$

Organizing Eq. 5.6 gives Proposition 5.1, completing the proof. □

5.3.5 Prediction length

The prediction length $\tau_{d,k}$ is the age of the measured data. The prediction length is on-line measured using built-in timers in the processing resources. Because the sensing delay is typically larger than the actuation period, there are cases when the predictor block does not have new sensing information to predict the state vector. Therefore, the same sensing information can be used to estimate the state vector more than once if the prediction length is updated. For example, in the serial case of Fig. 5.4a, the first discrete-time delay corresponds to four actuation periods, which means that a sensing operation is completed before time $k = 4$. The sensed information corresponds to a sample captured at time $k = 0$. Therefore the prediction length is $\tau_{d,4} = 4$ actuation periods. In the next time step, $k = 5$, no new sensing information is available. Therefore the prediction length is updated to $\tau_{d,5} = 5$ actuation periods. The variation of $\tau_{d,k}$ at run-time for the running example is summarized in Table 5.1. The prediction length varies from 2 to 7 actuation periods in the serial case while in the pipelined case it ranges from 2 to 5 actuation periods. The bounds of the prediction length for a lower bound $\underline{\tau}$ and an upper bound τ on the sensing delay are defined as:

$$\underline{\tau}_d = \left\lceil \frac{\underline{\tau}}{h_p} \right\rceil \tag{5.7}$$

$$\overline{\tau}_d = \left\lceil \frac{\tau}{h_p} \right\rceil + \left\lceil \left\lceil \frac{\tau}{h_p} \right\rceil \frac{1}{\gamma} \right\rceil - 1, \tag{5.8}$$

with $\underline{\tau}_d \in \mathbb{Z}^+$ the discrete-time lower bound of the prediction length, $\overline{\tau}_d \in \mathbb{Z}^+$ the upper bound of the prediction length, and $\gamma \in \mathbb{Z}^+$ the number of processing resources. Eqs. 5.7 and 5.8 show that adding more sensing resources (i.e., increasing γ) only affects $\overline{\tau}_d$.

Further, to compare the upper bound of the prediction length of controllers with different actuation periods, we use

$$\overline{\tau}_c = \overline{\tau}_d * h_p, \tag{5.9}$$

with $\overline{\tau}_c \in \mathbb{R}^+$ the upper bound of the prediction length in seconds.

5.3.6 Impact of prediction quality on QoC and control robustness

In an ideal case, the model-based predictions perfectly estimate the state vector after the variable delay. However, in practice it is common to have a margin of error in the plant model (i.e., to have model uncertainties), which affects the quality

Table 5.1: Prediction length in terms of actuation periods in the configurations of Fig. 5.4

current time k	last sensed information time		prediction length $\tau_{d,k}$	
	serial	pipelined	serial	pipelined
1	X	X	X	X
2	X	X	X	X
3	X	X	X	X
4	0	0	4	4
5	0	0	5	5
6	0	2	6	4
7	0	2	7	5
8	4	4	4	4
9	4	4	5	5
10	8	6,8	2	2
11	8	6,8	3	3
12	10	10	2	2
13	10	11	3	2

of the predictions. Since the predictions are used by the controller to compute the control input, low-quality predictions have a negative impact on the controller QoC and ultimately its robustness. The prediction length plays also a role in the prediction quality: long predictions rely more on the model and control inputs than on the measurements. Proposition 5.1 shows that with a long prediction (i.e., $\tau_{d,k} \to \infty$) the measurement term $(A_d)^{\tau_{d,k}} x_{k-\tau_{d,k}} \to 0$ because the spectral radius of A_d is less than 1, while the second term has always several larger terms given by the sum of elements $(A_d)^{\tau_{d,k}-1-j} B_d u_{k-\tau_{d,k}+j}$. Model uncertainties have therefore a strong negative impact in long predictions, much stronger than in short predictions. Since this strong negative impact in turn affects the QoC and control robustness, a limitation on the prediction length needs to be established.

5.3.7 QoC benchmarking

For the QoC benchmarking of the variable-delay controller, we include model uncertainties in the dynamic model:

$$\dot{x}(t) = (A_c + \Delta A_c)x(t) + (B_c + \Delta B_c)u(t), \tag{5.10}$$

with $\Delta A_c \in \mathbb{A}_c \subseteq \mathbb{R}^{n\times n}$ and $\Delta B_c \in \mathbb{B}_c \subseteq \mathbb{R}^{n\times 1}$ the uncertainties of the state and input matrices, respectively. The uncertainties can be obtained from the tolerance of each element of the matrices A_c and B_c.

Remark 5.1. *Chapter 3 required to have a particular structure in the uncertainties of Eq. 5.10, because of the discretization procedure applied in a later stage. In this chapter, such a discretization is not necessary. Therefore the particular structure on the uncertainties is no longer necessary.*

5.4 Variable-delay control design

The procedure for designing a variable-delay controller is divided into the following four subsections: histogram characterization, selection of candidate actuation periods, control design, and QoC benchmarking.

5.4.1 Histogram characterization

We select a set of actuation periods that potentially improve QoC. An actuation period similar to the worst-case delay, i.e., $h_p \to \tau$ produces a QoC similar to a worst-case design. The benefit of using a variable-delay control materializes when the actuation period is chosen significantly smaller than the worst-case delay. Ideally, a short actuation period $h_p \to 0$ is preferred because the resulting QoC is similar to the continuous-time controller with no delay. However, a small actuation period causes a long $\tau_{d,k}$, which is strongly affected by model uncertainties (see Section 5.3.6). The controller actuation periods are therefore chosen taking into account the most common delays shown in the implementation-dependent delay histogram. To do so, we follow the following procedure:

1. **Identification of the modes in the delay histogram:** Using the delay histogram, we define a set $\mathbb{M}$ with the most commonly occurring delays (i.e., delay modes) in the histogram:
$$\mathbb{M} = \{\tau_1, \tau_2, \ldots, \tau_M\},$$

with $\tau_1, \ldots, \tau_M$ the delay modes. We only consider modes that are significantly shorter than the worst-case delay because an actuation period selected based on a mode near the worst case potentially produces little or no improvement in control performance.

Example 5.1. ***Histogram-mode identification:*** *Consider the delay histogram presented in Fig. 5.2. The set with the histogram modes is given by* $\mathbb{M} = \{51, 102\}$. *All times are given in* ms.

2. **Selection of representative delays:** Using the histogram modes of the set $\mathbb{M}$, we define a set $\mathbb{D}$ with representative delays as:

$$\mathbb{D} = \{d_1, d_2, \ldots, d_d\},$$

with $d_1, \ldots, d_d$ the representative delays. Each representative delay covers (i.e., it is larger than) at least one mode. If two modes are close to each other, they can be covered by one representative delay.

Example 5.2. ***Representative-delay selection:*** *Continuing with Example 5.1, the set of representative delays is chosen to be* $\mathbb{D} = \{60, 120\}$. *Note that we intentionally choose delays that are slightly larger than the modes, so that the delays cover the full body of the peaks occurring around the modes.*

These two steps can be done manually with limited effort. They can also be automated using the scenario-identification technique presented in [45].

5.4.2 Selection of candidate actuation periods and resource configuration

Using the representative delays $\mathbb{D}$, we define a set $\mathbb{H}$ with candidate actuation periods as:

$$\mathbb{H} = \left\{h_{p1}, h_{p2}, \ldots, h_{ph}\right\},$$

with $h_{p1}, \ldots, h_{ph}$ the actuation periods. The elements of the set $\mathbb{H}$ and the resource configuration γ are defined to guarantee a desired QoC and robustness. We follow these steps:

1. **Actuation-period selection:** We select the actuation periods to guarantee performance improvement when the representative delays occur in the platform. Rearranging Eq. 5.7 and replacing $\underline{\tau}$ with all values of $\mathbb{D}$, the actuation periods are defined as

$$\mathbb{H} = \left\{d_i / \underline{\tau}_d \;\middle|\; d_i \in \mathbb{D}\right\},$$

with $\underline{\tau}_d$ the desired minimum prediction length. $\underline{\tau}_d$ is a design parameter chosen based on the desired QoC and the model reliability (i.e., the amount of uncertainties in the model). For example, a large $\underline{\tau}_d$ strongly enhances the QoC when a highly reliable model (i.e., a model with low uncertainties) is available; a small $\underline{\tau}_d$ can still enhance the control performance when a less reliable model is available. More than one value of $\underline{\tau}_d$ can be considered for each actuation period in the design exploration. The elements of the set $\mathbb{D}$ are used because these are the most common delays appearing in the platform.

Example 5.3. ***Actuation-period selection:*** *Continuing with Example 5.2, we consider* 10% *uncertainties in* A_c. *Since there is a relatively high uncertainties, we choose* $\underline{\tau}_d \in \{1,2\}$, *which results in actuation periods* $\{60,120\}$ *and* $\{30,60\}$, *respectively. Therefore, we take* $\mathbb{H} = \{30,60,120\}$.

2. **Resource-configuration selection:** We select the resource configuration to guarantee control robustness. As described in Section 5.3.6, long prediction lengths have a strong negative impact on prediction quality and control robustness. Therefore, to guarantee control robustness we, limit the prediction length in time $\overline{\tau}_c$ to a desired value. To do so, for each actuation period in the set $\mathbb{H}$ we use Eqs. 5.8 and 5.9 with τ equal to the maximal delay d_d in $\mathbb{D}$ to select a γ such that $\overline{\tau}_c$ stays within a desired bound. d_d replaces τ in Eq. 5.8 since it is the largest among the selected delays, which gives the longest $\overline{\tau}_c$, while τ is considered unlikely to happen. $\overline{\tau}_c$ is used because it allows to compare multiple actuation periods. If the resulting $\overline{\tau}_c$ is longer than desired and not enough processing resources are available, the designer has to select a shorter $\underline{\tau}_d$ in the previous step (i.e., one has to accept a lower performance).

Example 5.4. ***Resource-configuration selection:*** *Continuing with Example 5.3, with* $h_p = 30$ ms *and* $d_2 = 120$ ms, *we aim to have* $\overline{\tau}_c \leq 250$ ms. *Using* $\gamma = 1$ *in Eqs. 5.8 and 5.9 gives* $\overline{\tau}_d = 7$ *and* $\overline{\tau}_c = 210$ ms, *meeting the design requirement.*

5.4.3 Control design

Using the set of candidate actuation periods $\mathbb{H}$, one controller is designed per actuation period. Because the uncertainties on the model might produce steady-state errors, a controller with integral action is preferred. Compared with the control-design strategies of Chapters 3 and 4, no switching behaviour is required in the control design. Therefore, a single step LQI design is used to improve QoC in terms of time-domain metrics.

To include the integral action in the controller, the continuous-time model of Eq. 5.1 is discretized using Eq. 5.3. Then, the discrete-time model is augmented as follows:

$$\hat{z}_{i,k+1} = A_i \hat{z}_{i,k} + B_i u_k, \tag{5.11}$$

with $\hat{z}_{i,k} \in \mathbb{R}^{(n+1)\times 1}$ the augmented integral state vector, $A_i \in \mathbb{R}^{(n+1)\times(n+1)}$ and $B_i \in \mathbb{R}^{(n+1)\times 1}$ the augmented integral state and input matrices, respectively. These matrices and vector are further defined by

$$\hat{z}_{i,k} = \begin{bmatrix} \hat{x}_k \\ \hat{x}_{i,k} \end{bmatrix}, A_i = \begin{bmatrix} A_d & 0 \\ C_c & 1 \end{bmatrix}, B_i = \begin{bmatrix} B_d \\ 0 \end{bmatrix}.$$

$\hat{x}_{i,k} \in \mathbb{R}$ is the augmented state for the integral action computed from

$$\hat{x}_{i,k+1} = C_c \hat{x}_k + \hat{x}_{i,k} - r, \tag{5.12}$$

with $r \in \mathbb{R}$ the reference. The integral action (i.e., the convergence of the output to a constant reference) is straightforwardly demonstrated finding the steady-state value of y_k [14].

The control law with integral action is then defined as:

$$u_k = K\hat{x}_k + K_i \hat{x}_{i,k}, \tag{5.13}$$

where $K \in \mathbb{R}^{1\times n}$ is the feedback gain and $K_i \in \mathbb{R}$ is the integral gain. The control law is re-written in a matrix notation as:

$$u_k = K_{aug} \hat{z}_{i,k} \tag{5.14}$$

with $K_{aug} = \begin{bmatrix} K & K_i \end{bmatrix}$.

To design K_{aug}, we use the well-known LQR controller that includes integral action [21]. Such a controller trades off control effort with state deviation by minimizing the following cost function:

$$J = \sum_{k=0}^{\infty} \left(\hat{z}_{i,k}^T Q_i \hat{z}_{i,k} + u_k^T R_i u_k \right), \tag{5.15}$$

where $Q_i \in \mathbb{R}^{(n+1)\times(n+1)}$ and $R_i \in \mathbb{R}^{1\times 1}$ are the state and input weight matrices of the LQR, which are tuning parameters. To find the values of Q_i and R_i that produce a controller with minimum settling time, we use the PSO algorithm detailed in Section 2.5 with updated control law (i.e., Eq. 5.14) and updated plant model (Eq. 5.11). The resulting controller K_{aug} contains the feedback gain K and the integral gain K_i that are used by the variable-delay controller.

Example 5.5. ***Controller design:*** *Consider the dynamic model presented in Section 2.2:*

$$A_c = \begin{bmatrix} 0 & 1.7 \\ -9 & -2.5 \end{bmatrix}, B_c = \begin{bmatrix} 0 \\ 10 \end{bmatrix}, C_c = [1\ 0], x = \begin{bmatrix} x^1 \\ x^2 \end{bmatrix}.$$

For the purpose of this example, we only use one of the actuation periods in $\mathbb{H}$, namely the one that ultimately gives the best performance in terms of QoC. However, notice that in a full design exploration, one controller per element in $\mathbb{H}$ needs to be designed. Using the actuation period of Example 5.4, $h_p = 30$ ms, the discrete-time model is found using Eq. 5.3:

$$A_d = \begin{bmatrix} 0.99 & 0.04 \\ -0.25 & 0.92 \end{bmatrix}, B_d = \begin{bmatrix} 0.75 \\ 28.84 \end{bmatrix} \times 10^{-2}.$$

The augmented model is then defined by:

$$\hat{z}_i = \begin{bmatrix} \hat{x}^1 \\ \hat{x}^2 \\ \hat{x}_i \end{bmatrix}, A_i = \begin{bmatrix} 0.99 & 0.04 & 0 \\ -0.25 & 0.92 & 0 \\ 1 & 0 & 1 \end{bmatrix}, B_i = \begin{bmatrix} 0.75 \\ 28.84 \\ 0 \end{bmatrix} \times 10^{-2}.$$

with $\hat{x}_{i,k+1} = C_c\hat{x}_k + \hat{x}_{i,k} - r$. Applying the control-design strategy of Chapter 2, gives an augmented gain:

$$K_{aug} = \begin{bmatrix} -177.04 & -5.85 & -72.38 \end{bmatrix}.$$

The gains of the controller law are then defined as $K = \begin{bmatrix} -177.04 & -5.85 \end{bmatrix}$ and $K_i = -72.38$.

5.4.4 QoC benchmarking to select the best controller

As part of the design process, simulations are used to benchmark the QoC of the designed controllers. Each candidate controller is tested using the continuous-time model with uncertainties of Eq. 5.10. One simulation is run with each delay of the set $\mathbb{D}$. The delay is kept constant during the entire simulation. This procedure gives bounds on the performance of the controller: the best-case performance is expected with d_1 (i.e., the smallest element in $\mathbb{D}$) while the worst-case performance is expected with d_d (i.e., the largest element in $\mathbb{D}$). The average performance is influenced by the probability of occurrence of each delay in practice. In a run-time simulation, the QoC typically varies between the QoC obtained with benchmarked delays, because those are chosen to cover (i.e., are slightly larger than) the most commonly-occurring delays in the platform.

To compare if there is any performance gain in using the set of variable-delay controllers, a controller with a constant worst-case delay is designed using the procedure

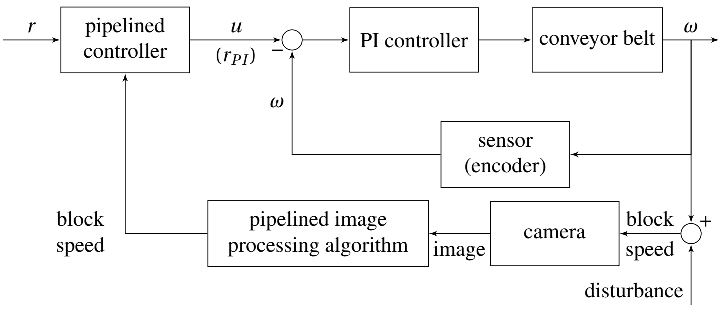


Figure 5.5: Controller structure of one of the belts in the case study.

of Chapter 2. The worst-case controller is also tested with the model with uncertainties shown in Eq. 5.10. The QoC of both control strategies is compared. We select the actuation periods which show potential QoC improvement compared to the worst-case design. If no improvement is achieved with the selected actuation periods or the resulting controllers are potentially unstable, the procedure of Section 5.4.2 has to be repeated with a smaller $\underline{\tau}_d$ or with a larger γ. A complete example of QoC benchmarking is provided in Section 5.6.

5.5 Case study: xCPS

We have introduced the design of a variable-delay controller based on a delay histogram. We now present a motivational set-up that shows the benefit of a variable-delay control over a worst-case-based control. We use the assembly line xCPS introduced in Section 2.2. We summarize the operation of xCPS and the need for a DISC.

xCPS is an assembly line that puts together circular complementary blocks [2]. xCPS is equipped with conveyor belts that transport the blocks through multiple actuators to complete the assembly process. The block speed needs to be regulated in the assembly process. A camera and an image-processing algorithm are used as sensor to detect block speed. An IBC can then regulate the block speed by changing the input to the belts. A better QoC (i.e., shorter settling time) means that the blocks move faster between actuators. Consequently, they are assembled faster, which potentially improves the productivity of xCPS.

A schematic of the controller regulating the conveyor belts is shown in Fig. 5.5. A continuous-time controller regulates the speed of rotation of the electric motors that drive the conveyor belts. The continuous-time controller does not have information about the speed or location of the block travelling on the belt. Ideally the motor speed and the block speed should be the same. However due to non-modelled effects (e.g., block friction with the borders of the belt) they may differ. A camera and an image-based processing algorithm are used as sensor to detect the positions and speeds of the blocks travelling on the conveyor belt. A DISC (i.e., an IBC) can regulate the speed of the assembly blocks on the belts by changing the input to the continuous-time PI controller. The plant to be controlled by the IBC is composed of the continuous-time controller and the DC motor model.

5.5.1 Plant model

The (nominal) model of the continuous-time controller and the DC motor is given by

$$A_c = \begin{bmatrix} -41.59 & 27.62 \\ -1 & 0 \end{bmatrix}, B_c = \begin{bmatrix} 0.69 \\ 1 \end{bmatrix},$$

where the states are $\begin{bmatrix} x^1(t) & x^2(t) \end{bmatrix}^T = \begin{bmatrix} \omega(t) & \int (r_{PI}(t) - \omega(t))dt \end{bmatrix}^T$, $\omega(t)$ is the block angular speed, and $r_{PI}(t)$ is the reference sent to the continuous-time controller. The camera and the image-processing algorithm are used to measure $\omega(t)$ from the assembly blocks.

We consider that the values on the state matrix can vary up to 15% of their nominal value. No uncertainties are considered in the input matrix. Therefore, the maximum uncertainty matrices of Eq. 5.10 are given by

$$\Delta A_c = \begin{bmatrix} -6.23 & 4.14 \\ -0.15 & 0 \end{bmatrix}, \Delta B_c = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

5.5.2 Image-processing algorithm

The image-processing algorithm on xCPS uses the Hough transform for circles to detect the block positions and speeds. We consider the delay histogram shown in Fig. 5.2 for the image-processing algorithm. In the histogram, there are two dominant modes produced by the number of blocks being detected in the image: the first mode corresponds to one block, while the second mode corresponds to two blocks. The best-case and worst-case execution times of the image-processing algorithm are given by $\underline{\tau} = 45\ ms$ and $\tau = 160\ ms$, respectively. The worst-case delay is significantly larger than the histogram modes. The worst-case delay is produced by a combination of algorithm-related variations (e.g., amount of features to process in the image) and hardware-related variations (e.g., cache misses). A variable-delay controller potentially improves the performance compared to a worst-case design. Two processing resources are available for executing the image-processing algorithm and the IBC.

5.6 Simulation results

We use the procedure of Section 5.4 to design a variable-delay controller with multiple resource configurations.

5.6.1 Histogram characterization

Using the procedure of Section 5.4.1 and Fig. 5.2, we identify the histogram modes as $\mathbb{T} = \{51, 102\}$. Next, we select the representative delays as $\mathbb{M} = \{60, 120\}$ to cover the histogram modes.

5.6.2 Selection of actuation periods and resource configuration

Using the procedure of Section 5.4.2 and the 15% expected uncertainties described in Section 5.5.1, we choose to have lower bounds on the prediction length of $\underline{\tau}_d = \{1, 2, 4\}$, resulting in actuation periods $\{60, 120\}$, $\{30, 60\}$, and $\{15, 30\}$, respectively. We then take

$$\mathbb{H} = \{15, 30, 60, 120\}.$$

These actuation periods potentially improve the control QoC because they are smaller than a worst-case based sampling period $h = 160\ ms$.

To select the resource configuration, the maximum prediction length is set as $\overline{\tau}_c \leq 180\ ms$. Considering $\gamma = 1$, Eqs. 5.8 and 5.9 result in $\overline{\tau}_c = \{225, 210, 180, 120\}$ per element of $\mathbb{H}$. Only the actuation periods $\mathbb{H}_1 = \{60, 120\}$ meet the prediction constraint. Considering $\gamma = 2$, $\overline{\tau}_c = \{165, 150, 120\}$ for $\mathbb{H}_2 = \{15, 30, 60\}$; $h_p = 120\ ms$ is not considered because that actuation period is larger than a worst-case actuation period $h = 80\ ms$ with $\gamma = 2$. In this case all actuation periods meet the minimum prediction-length constraint. Clearly, $\gamma = 2$ is then the preferred resource configuration, because this will result in the best QoC. However, we evaluate both configurations with all the actuation periods for demonstration purposes.

5.6.3 Control design

Using the control-design strategy of Section 5.4.3, the PSO algorithm is used to design one controller for each actuation period of the set $\mathbb{H}$. Likewise, to compare whether there is any performance gain, a constant worst-case control is also designed using the

PSO of Chapter 2. For the worst-case implementation the actuation period is chosen as $h = 160\ ms$ and $h = 80\ ms$ for $\gamma = 1$ and $\gamma = 2$, respectively (See Eq. 2.5).

For example using $h_p = 15\ ms$, the model from Section 5.5.1 is discretized as

$$A_i = \begin{bmatrix} 0.53 & 0.30 & 0 \\ -0.01 & 0.99 & 0 \\ 1 & 0 & 1 \end{bmatrix}, B_i = \begin{bmatrix} 0.01 \\ 0.01 \\ 0 \end{bmatrix}. \tag{5.16}$$

The augmented state vector is then $\hat{z}_{i,k} = \begin{bmatrix} \omega_k & \left(\int (r_{PI}(t) - \omega(t))dt\right)_k & \hat{x}_{i,k} \end{bmatrix}^T$. Note that the state $\left(\int (r_{PI}(t) - \omega(t))dt\right)_k$ represents the k^{th} sample of the continuous-time integral state. $\hat{x}_{i,k}$ is derived using Eq. 5.12:

$$\hat{x}_{i,k+1} = \hat{\omega}_k + \hat{x}_{i,k} - r.$$

The PSO algorithm of Chapter 2 is used to find the controller gains:

$$K_{aug} = \begin{bmatrix} -147.28 & -30.35 & -95.63 \end{bmatrix}.$$

The gains for the control law are then extracted as

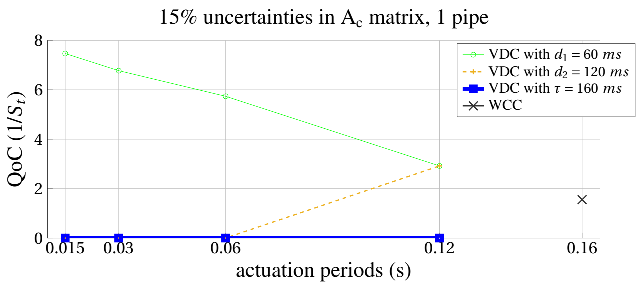
$$K = \begin{bmatrix} -147.28 & -30.35 \end{bmatrix}, K_i = -95.63.$$

5.6.4 QoC benchmarking

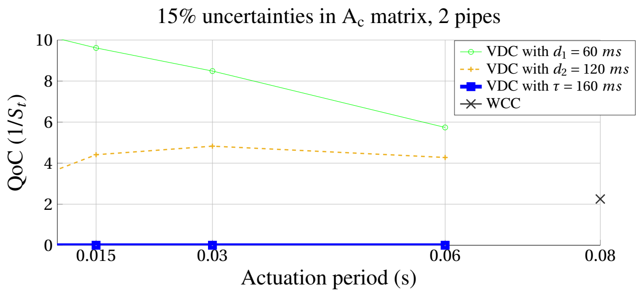
Using the procedure of Section 5.4.4, the QoC of the afore-designed controllers is benchmarked. For demonstration purposes, we also included τ (worst-case delay) in the benchmarked delays of the set $\mathbb{D}$. To find the QoC, the settling time is measured when the reference is changed. If the system output does not stabilize within a 2% margin, the QoC is set to zero (i.e., the system output keeps oscillating around the reference).

The benchmark is shown in Fig. 5.6a for $\gamma = 1$ and in Fig. 5.6b for $\gamma = 2$. In both configurations, the best performance is always achieved with the shortest representative delay $d_1 = 60\ ms$, because it gives the shortest prediction length. Increasing the delay deteriorates the QoC for all actuation periods. The evaluated controllers do not settle when τ is evaluated. However, Fig. 5.2 shows that the worst-case delay is unlikely to continuously occur. Therefore, the line drawn by d_2 in Fig. 5.6 can be considered as an average lower bound on QoC. In practice, the sensing delay will take any value between $\underline{\tau}$ and τ. However, the average QoC will be bounded between the QoC values for d_1 and d_2.

With $\gamma = 1$, the QoC with the actuation periods 15, 30, and 60 ms is zero when $d_2 = 0.12\ s$ is considered. Note that the actuation periods 15 and 30 ms do not meet



(a) Serial variable-delay control with $\gamma = 1$.

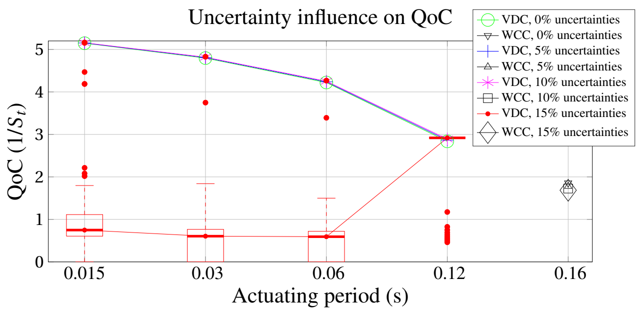


(b) Pipelined variable-delay control with $\gamma = 2$.

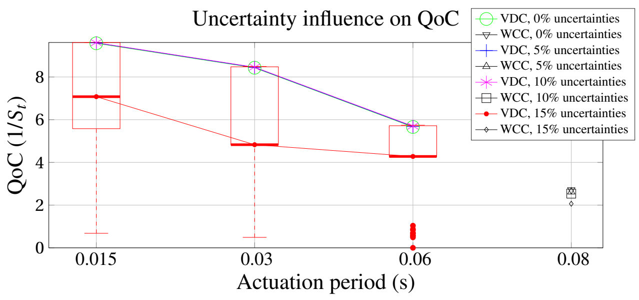
Figure 5.6: QoC benchmarking of variable-delay controllers (VDC) and the worst-case control design (WCC) using constant delays d_1, d_2, and τ.

the prediction-length constraint established in Section 5.6.2, while $h_p = 60$ ms barely does. This implies that the controller cannot tolerate the uncertainties. These actuation periods are therefore not desirable in a final implementation. $h_p = 120$ ms meets the prediction-length constraint and shows a QoC improvement with d_2 compared to a worst-case design. $h_p = 120$ ms is preferable in a final implementation. Note that with $h_p = 120$ ms, the performance with $d_1 = 60$ ms and $d_2 = 120$ ms is the same. Since both these delays are at most the sampling period of 120 ms, they require a prediction length of one sample, which in turn yields the same controller performance.

With $\gamma = 2$, the same actuation periods produce a better QoC than in a serial implementation. Additionally, in the pipelined case, all the actuation periods show improvement compared to a worst-case design. The best performing controller is the variable-delay controller with a representative delay $d_1 = 60$ ms. The recommended actuation period for this pipelined controller is $h_p = 15$ ms because it shows the best



(a) Serial variable-delay control.



(b) Pipelined variable-delay control.

Figure 5.7: Performance with 100 different delay sequences. The lines connect the median performance. The lines of the variable delay with 0% to 10% uncertainties are overlapping. A box plot is drawn for the largest uncertainties.

QoC improvement.

To evaluate the feasibility of the approach, the variable-delay controllers are also simulated on a realistic scenario with model uncertainties and varying-delay sequences. Figs. 5.7a and 5.7b show an average QoC of 100 simulations using the candidate sampling periods of Section 5.6.2, the worst-case design, multiple uncertainties in the A_c matrix, and variable-delay sequences generated according to Fig. 5.2. For the plot with 15% uncertainties, we use a box plot. The boxes correspond to 25 and 75 percentiles. The whiskers extend to the most extreme data points not considered as outliers, and the outliers are plotted individually using a red dot. Both graphs also show the effect of lower uncertainties between 0% and 10% on the QoC. For simplicity, only the median of the 100 simulations is plotted with these uncertainties. For uncertainties between

0% and 10%, the QoC deterioration is almost negligible, which means that the smallest element of the set $\mathbb{H}$ can be used.

Fig. 5.7a shows that for $\gamma = 1$, the resulting median QoC with 15% uncertainties is better than a worst-case design only with $h_p = 120\ ms$. The median QoC with $h_p \in \{15, 30, 60\}\ ms$ is close to zero, which makes them undesirable in a final implementation.

Fig. 5.7b shows that for $\gamma = 2$, the best QoC is achieved with the smallest actuation period and the smallest representative delay, outperforming a worst-case design. The median QoC is also between the bounds presented in Fig. 5.6b. Note that the lower whisker extends to a QoC which is below the worst-case design. This occurs because some delay sequences contained values close to τ which extends the settling time of the controller.

We tried to compare the QoC of our controllers with the predictor approach of [46]. This approach considers $\tau_{d,k}$ constant in the predictor of Section 5.3.3 (i.e., $\tau_{d,k} = \tau_L$ where $\tau_L \in \mathbb{R}^+$ is a constant prediction horizon). Using LMIs, the stability is verified against prediction-length errors (i.e. when $\tau_L \neq \tau_{d,k}$) in Theorem 1 as well as against prediction-length errors combined with model uncertainties in Theorem 2 [46]. In our approach, prediction-length errors are not present since $\tau_{d,k}$ is on-line measured and updated at every prediction; the stability against model uncertainties is addressed using the benchmark presented in Section 5.4.4. To compare QoC of both approaches, we attempted to solve the LMI of Theorem 1 of [46] for our case study (i.e. our controllers, delay range, and multiple values of τ_L). For the considered range of delays, Theorem 1 does not yield any feasible stable solution for the controller. This is mainly because of the conservative nature of the analysis for dealing with lack of information on the delay behaviour.

5.7 Summary

In this chapter, we have presented an implementation-aware variable-delay pipelined-control design strategy for DISC. Our controller compensates for the variable nature of the delay using a predictor, which estimates the state vector using a system model, the delayed state-vector measurement, and on-line measurements of the sensing delay. We presented a simulation benchmark that uses the implementation-dependent delay histograms and model uncertainties to select a resource configuration and a controller that improves performance compared to a constant worst-case design.

Our simulation results show that in a system without uncertainties, the shortest actuation period yields the best control performance and robustness. However, uncertainties have a strong negative impact on quality of the predictions, much stronger with a longer prediction length, which in turn affects the control performance. The prediction length can be decreased by increasing the actuation period or adding processing resources in a pipelined fashion. Extra sensing resources improve the control performance and the controller robustness.

Chapter 6

Conclusions and future work

6.1 Conclusions 134
6.2 Future work 137

Cyber-Physical Systems (CPSs) are becoming common in various domains including flexible manufacturing systems, automotive systems, aero-space systems, robotic systems, and many others. Common design specifications for CPSs are expressed in terms of performance, reliability, and cost-effectiveness. Designing a CPS involves a close interaction between multiple engineering disciplines. For example, the embedded-systems and the control-systems domains are combined to co-design embedded control systems. Embedded control systems are integrated into CPSs to regulate the physical behaviour of components in the system. The growing capabilities of embedded systems enabled a new generation of embedded control systems: the Data-Intensive Sensing Controllers (DISCs). DISCs use signal-processing algorithms to obtain sensing information for which no simple sensor exists. This sensing strategy extends the range of applications that can be regulated by embedded control systems. However, signal-processing algorithms require additional processing resources (for the signal-processing) which can negatively impact the cost-effectiveness of the CPS. Likewise, signal-processing algorithms introduce sensing delay into the control loop, which translates to limitations in the controller such as low performance (i.e., low Quality of Control (QoC)), robustness problems, and even control instability. Such limitations can have a direct impact on CPS specifications such as performance and reliability. This thesis contributes to the development of DISCs taking into account

CPS-level specifications.

Strategies to design DISCs are found in the embedded-systems domain as well as in the control-systems domain. Embedded-systems approaches profit from embedded systems with parallel processing (i.e., multiprocessing) capabilities to implement the sensing algorithm in a parallel manner. As a result, the sensing delay is shortened which is used to design controllers with an improved QoC. However, parallelizing an algorithm is time consuming and it is not applicable to all algorithms. Control-systems approaches use model-based predictions to generate extra sensing information to improve the QoC. However, model-based predictions are strongly affected by model mismatches and unmodelled disturbances. As an alternative to these embedded-systems and control-systems approaches, in this thesis, we used pipelined-sensing control.

Pipelined-sensing control consists of using embedded systems with sufficient multiprocessing capabilities to compute the sensing information in a pipelined fashion. As a result, additional sensing information is produced (compared to single-core implementations), which is used to design controllers with a better QoC. Pipelined control is easier to implement than embedded-systems-oriented approaches, since it does not require a profound analysis of the sensing algorithm to achieve parallelization. Further, pipelined control relies less on the model-based predictions of control-systems approaches because the extra sensing information is generated by actual sensing operations rather than by a model.

This thesis is the first to contribute to the design of pipelined-sensing control using modern embedded systems with multiprocessing capabilities. The thesis develops model-based design techniques to address the control limitations (i.e., low QoC and robustness problems) resulting from the sensing delay. The proposed strategies trade off processing resources with QoC and robustness, which can be used to guarantee the design specifications of CPSs in terms of performance, reliability, and cost-effectiveness.

6.1 Conclusions

This thesis presented a set of design techniques for pipelined control systems that can be used to guarantee CPS specifications such as performance, reliability, and cost-effectiveness. We identify two development stages of the CPS: the *design stage* (Chapters 2 and 3) and the *running stage* (Chapters 4 and 5). The *design stage* refers to the phase when the DISC is being designed (or redesigned) and substantial changes to the number of processing resources are acceptable; the *running stage* refers to when the DISC is already operational, but its specifications must be improved with minimum changes in the number of processing resources. Note that the techniques of the *running stage* can also be applied at the *design stage*, as we elaborate in the paragraphs below.

For DISCs that are in the *design stage*, we presented several techniques to design pipelined control systems. These techniques can be classified according to the existence of model uncertainties. In case the model contains only negligible uncertainties, the design technique of Chapter 2 or the techniques of the *running stage* can be considered. The technique from Chapter 2 analyses the resources-QoC trade-off in pipelined-sensing control. We introduced this technique because in pipelined control the use of more processing resources results in a better QoC, creating the aforementioned trade-off. The result of this analysis allows to select a resource configuration that meaningfully improves the QoC. The selected resource configuration guarantees the cost-effectiveness of a CPS, while the improved QoC can be used to improve the CPS performance. In case the trade-off analysis suggests more processing resources than what is available for the controller, the techniques presented for the *running stage* can be considered. However, notice that the techniques of the *running stage* require additional knowledge of the embedded system, such as knowledge of the sporadic applications in the technique of Chapter 4 or knowledge of the histogram of delays in the technique of Chapter 5. These techniques are discussed in the paragraphs below.

In case the model contains significant uncertainties, the design technique of Chapter 3 (or the *running stage* technique of Chapter 5) can be applied. The technique of Chapter 3 extends the trade-off analysis of Chapter 2 to include model uncertainties. We introduced this technique because model uncertainties have a negative impact on QoC, which may affect system-level specifications. Such a technique helps to select a resource configuration that meaningfully improves the QoC while guaranteeing that the controller is robust against model uncertainties. As in the previous case, the selected resource configuration guarantees the cost-effectiveness of the CPS, the better QoC can be used to improve the CPS performance, and the guaranteed controller robustness can be used to satisfy CPS reliability. In case the model uncertainties are not significant (i.e., they do not produce a visible negative impact on the QoC), the method of Chapter 2 should be applied instead of the method of Chapter 3. This is suggested because the robustness analysis of Chapter 3 might (unnecessarily) constrain the QoC to guarantee stability in any situation, while adding complexity to the trade-off analysis. Note that the design technique of the *running stage* proposed in Chapter 5 can also be applied when the system contains significant uncertainties. A comparison between the applicability of these techniques is discussed below.

For DISCs that are in the *running stage*, we consider (like for the previous stage) two cases depending on the existence of model uncertainties. When the model has only negligible uncertainties, the technique of Chapter 4 can be used. The technique proposes a Reconfigurable Pipelined Controller (RPC), which allows to on-line change (i.e., reconfigure) idle processing resources from other operations to compute additional instances of the signal-processing algorithm. The extra sensing information is used to further improve the QoC. The use of this technique is therefore conditioned to the temporal availability of extra processing resources from other applications. RPC can also be used as an additional step of the trade-off analysis of Chapter 2 in case the suggested resource configuration is not statically available in the embedded system. From the system-level perspective, a better QoC can be used to improve the CPS performance. Additionally, the system cost-effectiveness is also improved because no additional processing resources are needed while the only new requirement is that the data-intensive sensor can cope with the changing sampling period of the RPC.

When the model contains uncertainties, the design technique of Chapter 5 can be used. This technique requires extra information about the signal-processing algorithm delay: the delay has to be characterized with the histograms of delays, and the delay has to be measured on-line after each sensing operation is finished. Using this extra information, a variable-delay pipelined control is designed to further improve the QoC. However, the QoC improvement depends on the model quality, e.g., a model with large uncertainties performs significantly worse than a model with small uncertainties. Therefore, the applicability of this technique is conditioned to the model quality and to the availability of the extra information about the delay. From the system-level viewpoint, the QoC improvement can be used to enhance the performance of the CPS. Since the processing resources are not increased, the CPS becomes more cost-effective. Likewise, since the control design takes into account model uncertainties, the CPS reliability can also be guaranteed with this strategy. Note that this technique can also be applied after the *design stage* provided that the model quality meets the assumption and that histograms of delays are available. However, this technique does not provide a trade-off analysis between processing resources and QoC, which might affect the cost-effectiveness of the CPS.

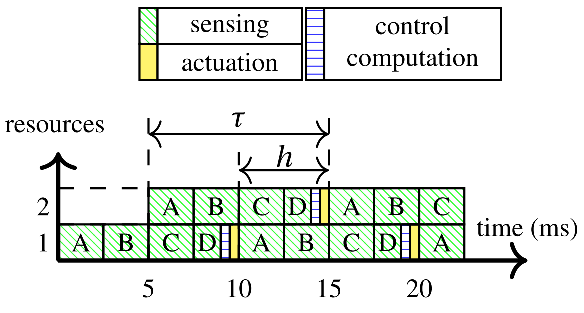
Finally, a comparison of the techniques of Chapter 5 with those of Chapter 3 shows that the former relies on model simulations to give the stability bounds of the controller, while the latter relies on a mathematical analysis to provide such stability bounds. Simulations (generally) provide less conservative stability bounds than a mathematical analysis. However, a mathematical analysis can be adapted faster to a wider range of applications. Therefore, the use of these techniques depends on the particular requirements of the CPS.

This thesis contributed in the design of pipelined-sensing controllers using multiprocessing embedded systems. The design strategies proposed in this thesis allow to design controllers that trade off processing resources with QoC and control robustness. These strategies can therefore be used in the design of CPS to improve on performance, reliability, and cost-effectiveness.

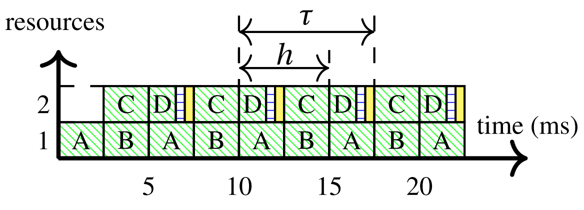
6.2 Future work

While this thesis has contributed substantially to the design of pipelined-sensing controllers using multiprocessing embedded systems for CPSs, our work can be further extended in the following directions:

- **Parallel-pipelined-sensing control:** Current pipelined-sensing control considers multiple serial signal-processing-algorithm instances implemented in a pipelined fashion, as Fig. 6.1a shows. However, in case parallelization is possible in the signal-processing algorithm, the algorithm can be implemented in a parallel pipelined fashion, which reduces the sensing delay and can be used to further improve the QoC. Fig. 6.1b shows an example of parallel pipelined control.



(a) Pipelined control.



(b) Parallel pipelined control.

Figure 6.1: Comparison of pipelined controller strategies. The signal-processing algorithm executes sub-tasks *A*, *B*, *C*, and *D*. Some sub-tasks can be executed concurrently. Using a parallel-pipelined sensing control the sensing delay τ is reduced compared to a pipelined implementation. Notice that the sampling period h remains the same.

Parallel pipelined control involves design challenges from the embedded-systems domain and from the control-systems domain. In the embedded-systems domain, algorithm parallelization is application-specific and time consuming. Such a parallelization requires an algorithm model to explore the parallel components of the algorithm, data dependencies, communication overheads, shared memory policies, load balancing, among others. Likewise, strategies to allocate the processing resources among the parallel components of the algorithm are necessary, such that the timing requirements of the control design are met (e.g., keeping a sampling period constant or bounded to a range). Existing algorithm-modelling techniques based on data-flow analysis can potentially be used to cope with these algorithm related challenges [115, 116]. However, combining such algorithm-modelling techniques with parallel pipelined control remains an open challenge.

In the control-systems domain, modelling techniques are required such that the interplay between the parallel-pipelined sensing and the plant model is captured. Likewise, control-design strategies are also required such that the QoC is optimized while trading off processing resources and control robustness. The control-design strategies presented in this thesis are a potential alternative to cope with these control challenges. However, adaptation to the new sensing strategy is still required as well as coping with the information provided by the algorithm-modelling technique.

- **State-observer pipelined-sensing control:** The current pipelined strategy assumes that the complete state vector is measured via the signal-processing algorithm. Although this is a valid assumption in many applications, there are some cases where part of the state vector is unavailable or are measured via sensors with no delay. Adapting the current strategy can increase the span of CPSs that can benefit from pipelined-sensing control. To do so, it is necessary to consider control-design strategies that compensate for the missing parts of the state vector while still optimizing the QoC. A potential solution is to use controllers with state observers or Linear Quadratic Gaussian (LQG) controllers (e.g., [77, 134]). However, these control-design strategies are strongly dependent on the plant model, which makes them vulnerable to model uncertainties. Additionally, these strategies commonly use QoC metrics related to control robustness or quadratic cost. Therefore, to apply these strategies in pipelined control, the effect of model uncertainties has to be further analysed while the QoC has to be measured in terms of real-time metrics.

- **Extension of robustness-analysis technique for pipelined-sensing control:** The robustness-analysis technique of Chapter 3 requires that the model uncertainties have a particular structure to perform the benchmarking of the discrete-time uncertainties. A strategy to analyse the robustness with no restriction in the uncertainties can increase the range of CPSs where the robustness-analysis technique for pipelined systems can be applied. However, the discretization of an uncertain matrix is a fundamental problem in control systems because of the non-linear nature of the discretization [131]. A potential solution for this challenge is to perform the robustness analysis in continuous time instead of discrete time. To do so, the modelling strategy of impulsive sampled-data systems (e.g., [98, 109] can be used to convert discrete-time components of the robustness analysis (i.e., the controller and the sensing delay) into continuous-time components. Next, a continuous-time robustness analysis technique can be applied using the original continuous-time uncertainties. This possibility remains to be explored for the pipelined-sensing control case.

Bibliography

[1] W. H. Aangenent, W. Heemels, M. van de Molengraft, D. Henrion, and M. Steinbuch, "Linear control of time-domain constrained systems," *Automatica*, vol. 48, no. 5, pp. 736–746, 2012.

[2] S. Adyanthaya, H. Alizadeh, J. Bastos, A. Behrouzian, R. Medina, J. van Pinxten, B. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, M. Hendriks, S. Stuijk, M. Reniers, and J. Voeten, "xCPS: A Tool to Explore Cyber Physical Systems," *SIGBED Rev.*, vol. 14, no. 1, pp. 81–95, 2017.

[3] S. Adyanthaya, H. Alizadeh, J. Bastos, A. Behrouzian, R. Medina, J. van Pinxten, B. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, S. Stuijk, M. Reniers, and J. Voeten, "xCPS: A Tool to eXplore Cyber Physical Systems," in *Proceedings of the WESE: Workshop on Embedded and Cyber-Physical Systems Education*. ACM, 2015, pp. 3:1–3:8.

[4] S. Adyanthaya, Z. Zhang, M. Geilen, J. Voeten, T. Basten, and R. Schiffelers, "Robustness analysis of multiprocessor schedules," in *Embedded Computer Systems: Architectures, Modeling and Simulation. IC-SAMOS 2014. International Conference on*, 2014, pp. 9–17.

[5] R. Agrawal, S. Gupta, J. Mukherjee, and R. K. Layek, "A GPU based Real-Time CUDA implementation for obtaining Visual Saliency," in *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*. ACM, 2014.

[6] P. Ajay, “Scenario-based switching of data-intensive controllers in Cyber-Physical System,” MSc thesis, Eindhoven University of Technology, 2016.

[7] S. Akaike, “On top of the world,” *Ansys advantage magazine*, vol. 9, no. 2, pp. 10–12, 2015.

[8] A. M. Annaswamy, D. Soudbakhsh, R. Schneider, D. Goswami, and S. Chakraborty, “Arbitrated network control systems: A co-design of control and platform for cyber-physical systems,” in *Control of Cyber-Physical Systems*. Springer, 2013, pp. 339–356.

[9] D. Antunes and W. P. M. H. Heemels, “Rollout event-triggered control: Beyond periodic control performance,” *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3296–3311, 2014.

[10] ARM, “ARM Cortex-R7 MPCore Technical Reference Manual,” ARM, Tech. Rep., 2014.

[11] W. F. Arnold and A. J. Laub, “Generalized eigenproblem algorithms and software for algebraic riccati equations,” *Proceedings of the IEEE*, vol. 72, no. 12, pp. 1746–1754, 1984.

[12] M. Bacic, “On hardware-in-the-loop simulation,” in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 3194–3198.

[13] T. Basten, J. Bastos, R. Medina, B. van der Sanden, M. C. W. Geilen, D. Goswami, M. A. Reniers, S. Stuijk, and J. P. M. Voeten, “Scenarios in the design of flexible manufacturing systems,” in *System-Scenario-based Design Principles and Applications*. Springer International Publishing, 2020, ch. 9, pp. 181–224.

[14] A. Bemporad, “Lecture notes in integral action in state feedback controls,” February 2010.

[15] S. Bittanti and P. Colaneri, *Periodic systems: filtering and control*. Springer Science & Business Media, 2008.

[16] S. Borkar, “Thousand core chips: A technology perspective,” in *Proceedings of the 44th Annual Design Automation Conference*, 2007, pp. 746–749.

[17] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear matrix inequalities in system and control theory*. SIAM, 1994, vol. 15.

[18] M. F. Braga, C. F. Morais, E. S. Tognetti, R. C. L. F. Oliveira, and P. L. D. Peres, "A new procedure for discretization and state feedback control of uncertain linear systems," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 2013, pp. 6397–6402.

[19] M. F. Braga, C. F. Morais, L. A. Maccari, E. S. Tognetti, V. F. Montagner, R. C. L. F. Oliveira, and P. L. D. Peres, "Robust stability analysis of grid-connected converters based on parameter-dependent lyapunov functions," *Journal of Control, Automation and Electrical Systems*, vol. 28, no. 2, pp. 159–170, 2017.

[20] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. CRC Press, 1975.

[21] S. Carrière, S. Caux, and M. Fadel, "Optimal LQI Synthesis for Speed Control of Synchronous Actuator under Load Inertia Variations," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 5831–5836, 2008.

[22] F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, pp. 82–90, 2006.

[23] S. Chroust, M. Vincze, R. Traxl, and P. Krautgartner, "Evaluation of processing architecture and control law on the performance of vision-based control systems," in *Proceedings of the international workshop in Advanced Motion Control (AMC)*. IEEE, 2000, pp. 19–24.

[24] M. B. G. Cloosterman, N. van de Wouw, W. P. M. H. Heemels, and H. Nijmeijer, "Stability of networked control systems with large delays," in *2007 46th IEEE Conference on Decision and Control*, 2007, pp. 5017–5022.

[25] M. Cloosterman, "Robust stability of networked control systems with time-varying network-induced delays," *Decision and Control, 2006 45th IEEE Conference on*, pp. 4980–4985, 2006.

[26] P. Colaneri, "Output stabilization via pole placement of discrete-time linear periodic systems," *IEEE Transactions on Automatic Control*, vol. 36, no. 6, 1991.

[27] G. Da Prato and A. Ichikawa, "Quadratic control for linear periodic systems," *Applied Mathematics and Optimization*, vol. 18, no. 1, pp. 39–66, 1988.

[28] J. Daafouz, P. Riedinger, and C. Iung, "Stability analysis and control synthesis for switched systems: a switched lyapunov function approach," *IEEE transactions on automatic control*, vol. 47, no. 11, pp. 1883–1887, 2002.

[29] M. Dehghan and M. H. Ang, “Stability of switched linear systems under dwell time switching with piece wise quadratic functions,” in *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, 2014, pp. 1257–1260.

[30] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, “Particle swarm optimization: basic concepts, variants and applications in power systems,” *Evolutionary Computation, IEEE Transctions on*, vol. 12, no. 2, pp. 171–195, 2008.

[31] L. Denissen, “Image-based control and throughput analysis forflexible manufacturing systems,” MSc thesis, Eindhoven University of Technology, 2016.

[32] P. J. Dhrymes, *Mathematics for econometrics*, 4th ed. Springer, 1978.

[33] R. C. Dorf and R. H. Bishop, *Modern control systems*. Pearson, 2011.

[34] D. Du, B. Jiang, and S. Zhou, “Delay-dependent robust stabilisation of uncertain discrete-time switched systems with time-varying state delay,” *International Journal of Systems Science*, vol. 39, no. 3, pp. 305–313, 2008.

[35] H. Duan and C. Sun, “Pendulum-like oscillation controller for micro aerial vehicle with ducted fan based on LQR and PSO,” *Science China Technological Sciences*, vol. 56, no. 2, pp. 423–429, 2013.

[36] Q. Fu, G. Xie, and L. Wang, “Stability Analysis and Stabilization Synthesis for Periodically Switched Linear Systems with Uncertainties,” *2005 American Control Conference*, pp. 30–35, 2005.

[37] H. Fujimoto, “Visual Servoing of 6 DOF Manipulator by Multirate Control with Depth Identification,” in *42nd IEEE International Conference on Decision and Control*, vol. 5. IEEE, 2003, pp. 5408–5413.

[38] H. Gao and T. Chen, “New results on stability of discrete-time systems with time-varying state delay,” *IEEE Transctions on Automatic Control*, vol. 52, no. 2, pp. 328–334, 2007.

[39] H. Gao, J. Lam, C. Wang, and Y. Wang, “Delay-dependent output-feedback stabilisation of discrete-time systems with time-varying state delay,” *IEE Proceedings-Control Theory and Applications*, vol. 151, no. 6, pp. 691–698, 2004.

[40] H. Gao and T. Chen, “New results on stability of discrete-time systems with time-varying state delay,” *IEEE Transactions on Automatic Control*, vol. 52, no. 2, pp. 328–334, 2007.

[41] P. Garcia, P. Castillo, R. Lozano, and P. Albertos, “Robustness with respect to delay uncertainties of a predictor-observer based discrete-time controller,” in *Decision and Control, 2006 45th IEEE Conference on*. IEEE, 2006, pp. 199–204.

[42] W. Geelen, D. Antunes, J. Voeten, R. Schiffelers, and M. Heemels, “The impact of deadline misses on the control performance of high-end motion control systems,” *IEEE Transctions on Industrial Electronics*, vol. 63, no. 2, pp. 1218–1229, 2016.

[43] D. Geer, “Chip makers turn to multicore processors,” *Computer*, vol. 38, no. 5, pp. 11–13, 2005.

[44] J. C. Geromel, M. C. d. Oliveira, and J. Bernussou, “Robust Filtering of Discrete-Time Linear Systems with Parameter Dependent Lyapunov Functions,” *SIAM Journal on Control and Optimization*, vol. 41, no. 3, pp. 700–711, 2002.

[45] S. V. Gheorghita, T. Basten, and H. Corporaal, “Profiling driven scenario detection and prediction for multimedia applications,” in *Embedded Computer Systems: Architectures, Modeling and Simulation. IC-SAMOS 2006. International Conference on*, 2006, pp. 63–70.

[46] A. Gonzalez, P. Garcia, P. Albertos, P. Castillo, and R. Lozano, “Robustness of a discrete-time predictor-based controller for time-varying measurement delay,” *Control Engineering Practice*, vol. 20, no. 2, pp. 102 – 110, 2012.

[47] A. Gonzalez, A. Sala, P. Garcia, and P. Albertos, “Robustness analysis of discrete predictor-based controllers for input-delay systems,” *International Journal of Systems Science*, vol. 44, no. 2, pp. 232–239, 2013.

[48] K. Goossens, M. Koedam, A. Nelson, S. Sinha, S. Goossens, Y. Li, G. Breaban, J. van Kampenhout, R. Tavakoli Najafabadi, J. Valencia, H. Ahmadi Balef, B. Akesson, S. Stuijk, M. Geilen, D. Goswami, and M. Nabi Najafabadi, “NoC-Based Multiprocessor Architecture for Mixed-Time-Criticality Applications,” *Handbook of Hardware/Software Codesign*, pp. 491–530, 2017.

[49] P. F. Gorder, “Multicore processors for science and engineering,” *Computing in Science Engineering*, vol. 9, no. 2, pp. 3–7, 2007.

[50] D. Goswami, R. Schneider, and S. Chakraborty, “Relaxing signal delay constraints in distributed embedded controllers,” *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2337–2345, 2014.

[51] J. K. Hale and S. M. V. Lunel, *Introduction to functional differential equations*. Springer Science & Business Media, 2013, vol. 99.

[52] J. Hamidi, “Control system design using particle swarm optimization (PSO),” *International Journal of Soft Computing and Engineering*, vol. 1, no. 6, pp. 116–119, 2012.

[53] J. Hansen, S. A. Hissam, and G. A. Moreno, “Statistical-based WCET estimation and validation,” in *Proceedings of the 9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.

[54] R. Hassan, B. Cohanim, O. De Weck, and G. Venter, “A comparison of particle swarm optimization and the genetic algorithm,” in *Proceedings of the 1st AIAA multidisciplinary design optimization specialist conference*, 2005, pp. 18–21.

[55] K. Hassani and W.-S. Lee, “Optimal tuning of linear quadratic regulators using quantum particle swarm optimization,” in *Procedings of the International Conference of Control, Dynamics and Robotics*, 2014, pp. 1–8.

[56] Y. He, M. Wu, G.-P. Liu, and J.-H. She, “Output feedback stabilization for a discrete-time system with a time-varying delay,” *IEEE Transactions on Automatic Control*, vol. 53, no. 10, pp. 2372–2377, 2008.

[57] S. Heath, “1 - what is an embedded system?” in *Embedded Systems Design (Second Edition)*, S. Heath, Ed. Newnes, 2002, pp. 1 – 14.

[58] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*. John Wiley & Sons, 2004.

[59] T. A. Henzinger and J. Sifakis, “The embedded systems design challenge,” in *Proceedings of the 14th International Conference on Formal Methods*. Springer-Verlag, 2006, pp. 1–15.

[60] V. Hernandez and L. Jodar, “Boundary problems and periodic Riccati equations,” *Automatic Control, IEEE Transactions on*, vol. 30, no. 11, pp. 1131–1133, 1985.

[61] L. Hetel, J. Daafouz, and C. Iung, “LMI control design for a class of exponential uncertain systems with application to network controlled switched systems,” in *American control conference*, 2007, pp. 1401–1406.

[62] N. J. Higham, *Functions of Matrices: Theory and Computation (Other Titles in Applied Mathematics)*. USA: Society for Industrial and Applied Mathematics, 2008.

[63] R. Hodrea, C. Ionescu, and R. De Keyser, “Predictive control strategy with on-line time delay estimation applied in general anaesthesia,” *IFAC Proceedings Volumes*, vol. 43, no. 2, pp. 253–258, 2010.

[64] *Industrial Cameras: 23/33/Z12/Z30 Series - Trigger and I/O*, The Imaging Source Europe GmbH, March 2017.

[65] R. Isermann, J. Schaffnit, and S. Sinsel, “Hardware-in-the-loop simulation for the design and testing of engine-control systems,” *Control Engineering Practice*, vol. 7, no. 5, pp. 643 – 653, 1999.

[66] K. Ito, B. M. Nguyen, Y. Wang, M. Odai, H. Ogawa, E. Takano, T. Inoue, M. Koyama, H. Fujimoto, and Y. Hori, “Fast and accurate vision-based positioning control employing multi-rate kalman filter,” in *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE*. IEEE, 2013, pp. 6466–6471.

[67] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, “Development of autonomous car—part I: Distributed system architecture and development process,” *Industrial Electronics, IEEE Transctions on*, vol. 61, no. 12, pp. 7131–7140, 2014.

[68] ——, “Development of autonomous car—part II: A case study on the implementation of an autonomous driving system based on distributed architecture,” *Industrial Electronics, IEEE Transctions on*, vol. 62, no. 8, pp. 5119–5132, 2015.

[69] A. Kawamura, K. Tahara, R. Kurazume, and T. Hasegawa, “Robust visual servoing for piece manipulation with large time-delays of visual information,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4797–4803.

[70] J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” in *Neural Networks, 1995. Proceedings ., IEEE International Conference on*, vol. 4, 1995, pp. 1942–1948 vol.4.

[71] S. Kestur, M. S. Park, J. Sabarad, D. Dantara, V. Narayanan, Y. Chen, and D. Khosla, “Emulating mammalian vision on reconfigurable hardware,” in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 141–148.

[72] S. K. Khaitan and J. D. McCalley, “Design techniques and applications of cyber-physical systems: A survey,” *IEEE Systems Journal*, vol. 9, no. 2, pp. 350–365, 2015.

[73] P. P. Khargonekar, I. R. Petersen, and K. Zhou, “Robust stabilization of uncertain linear systems: quadratic stabilizability and h $_\infty$ control theory,” *IEEE Transctions on Automatic Control*, vol. 35, no. 3, pp. 356–361, 1990.

[74] M. V. Kothare, V. Balakrishnan, and M. Morari, “Robust constrained model predictive control using linear matrix inequalities,” *Automatica*, vol. 32, no. 10, pp. 1361 – 1379, 1996.

[75] P. Krautgartner and M. Vincze, “Performance evaluation of vision-based control tasks,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 3. IEEE, 1998, pp. 2315–2320.

[76] ——, “Optimal image processing architecture for active vision systems,” in *Computer Vision Systems*, ser. Lecture Notes in Computer Science. Springer, 1999, vol. 1542, pp. 331–347.

[77] E. Lavretsky, “Adaptive output feedback design using asymptotic properties of LQG/LTR controllers,” *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1587–1591, 2012.

[78] M. A. C. Leandro, J. R. C. Júnior, and K. H. Kienitz, “Robust D-stability via discrete controllers for continuous time uncertain systems using LMIs with a scalar parameter,” in *Control and Automation (MED), 2015 23th Mediterranean Conference on*. IEEE, 2015, pp. 644–649.

[79] E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.

[80] F. L. Lewis and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 1995.

[81] H. Lin and P. Antsaklis, “Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results,” *IEEE Transactions on Automatic Control*, vol. 54, no. 2, pp. 308–322, 2009.

[82] B. Lincoln and B. Bernhardsson, “Optimal control over networks with long random delays,” in *Proceedings CD of the Fourteenth International Symposium on Mathematical Theory of Networks and Systems*. Laboratoire de Théorie des Systèmes (LTS), University of Perpignan, 2000.

[83] J. Löfberg, “Yalmip : A Toolbox for Modeling and Optimization in Matlab,” in *2004 IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2004.

[84] M. L. Loper, *Modeling and simulation in the systems engineering life cycle: core concepts and accompanying lectures*. Springer, 2015.

[85] R. Lozano, P. Castillo, P. Garcia, and A. Dzul, “Robust prediction-based control for unstable delay systems: Application to the yaw control of a mini-helicopter,” *Automatica*, vol. 40, no. 4, pp. 603 – 612, 2004.

[86] L. Lv, G. Duan, and B. Zhou, “Parametric pole assignment and robust pole assignment for discrete-time linear periodic systems,” *SIAM Journal on Control and Optimization*, vol. 48, no. 6, pp. 3975–3996, 2010.

[87] O. Mason and R. N. Shorten, “On common quadratic Lyapunov functions for stable discrete time LTI systems,” *IMA Journal of Applied Mathematics*, vol. 69, pp. 271–283, 2002.

[88] R. Medina, S. Stuijk, D. Goswami, and T. Basten, “Implementation-aware design of image-based control with on-line measurable variable-delay,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 240–245.

[89] R. Medina, S. Tabatabaei Nikkhah, D. Goswami, W. P. M. H. Heemels, S. Stuijk, and T. Basten, “Reconfigurable pipelined control systems,” *IEEE Design and Test*, pp. 1–1, 2020.

[90] R. Medina, S. Stuijk, D. Goswami, and T. Basten, “Reconfigurable pipelined sensing for image-based control,” in *Industrial Embedded Systems (SIES), 2016 11th IEEE Symposium on*. IEEE, 2016.

[91] ——, “Exploring the trade-off between processing resources and settling time in image-based control through LQR tuning,” in *Proceedings of the Symposium on Applied Computing*. ACM, 2017.

[92] R. Medina, J. Valencia, S. Stuijk, D. Goswami, and T. Basten, "Designing a controller with image-based pipelined sensing and additive uncertainties," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 3, pp. 33:1–33:26, 2019.

[93] R. Meyer and C. Burrus, "A unified analysis of multirate and periodically time-varying digital filters," *Circuits and Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 162–168, 1975.

[94] S. Mohamed, D. Goswami, V. Nathan, R. Rajappa, and T. Basten, "A scenario- and platform-aware design flow for image-based control systems," *Microprocessors and Microsystems*, vol. 75, p. 103037, 2020.

[95] M. M. Moldovan and M. S. Gowda, "On Common Linear/Quadratic Lyapunov Functions for Switched Linear Systems," in *Nonlinear Analysis and Variational Problems: In Honor of George Isac*. Springer New York, 2010, pp. 415–429.

[96] L. A. Montestruque and P. Antsaklis, "Stability of model-based networked control systems with time-varying transmission times," *IEEE Transctions on Automatic Control*, vol. 49, no. 9, pp. 1562–1572, 2004.

[97] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[98] P. Naghshtabrizi, J. Hespanha, and A. R. Teel, "Exponential stability of impulsive systems with application to uncertain sampled-data systems," *Systems and Control Letters*, vol. 57, no. 5, pp. 378 – 385, 2008.

[99] K.-E. Örzén and A. Cervin, "Control and embedded computing: Survey of research directions," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 191 – 202, 2005.

[100] S. Panda and N. P. Padhy, "Comparison of particle swarm optimization and genetic algorithm for facts-based controller design," *Applied Soft Computing*, vol. 8, pp. 1418 – 1427, 2008.

[101] A. Priyadi, N. Yorino, and Y. Zoka, "Design optimal feedback control using evolutionary particle swarm optimization in multi-machine power system," *Electrical Equipment Society National Japanesse Conference*, vol. 25, pp. 529–534, 2007.

[102] J. Qiu, G. Feng, and J. Y. J. Yang, "Delay-dependent robust H_∞ output feedback control for uncertain discrete-time switched systems with interval time-varying delay," *2008 American Control Conference*, pp. 3975–3980, 2008.

[103] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 731–736.

[104] K. J. Åström and B. Wittenmark, *Computer-controlled Systems (3rd Ed.)*. Prentice-Hall, Inc., 1997.

[105] J. Ren and Q. Zhang, "Robust h_{∞} control for uncertain descriptor systems by proportional–derivative state feedback," *International Journal of Control*, vol. 83, no. 1, pp. 89–96, 2010.

[106] I. Robandi, K. Nishimori, R. Nishimura, and N. Ishihara, "Optimal feedback control design using genetic algorithm in multimachine power system," *International Journal of Electrical Power & Energy Systems*, vol. 23, pp. 263 – 271, 2001.

[107] W. J. Rugh, *Linear system theory*. Prentice hall Upper Saddle River, NJ, 1996, vol. 2.

[108] T. Sanislav and L. Miclea, "Cyber-physical systems-concept, challenges and research areas," *Journal of Control Engineering and Applied Informatics*, vol. 14, no. 2, pp. 28–33, 2012.

[109] A. Seuret and C. Briat, "Stability analysis of uncertain sampled-data systems with incremental delay using looped-functionals," *Automatica*, vol. 55, pp. 274 – 278, 2015.

[110] P. Sharkey and D. Murray, "Delays versus performance of visually guided systems," *Control Theory and Applications, IEE Proceedings*, vol. 143, no. 5, pp. 436–447, 1996.

[111] L. S. Shieh, W. Wang, and G. Chen, "Discretization of cascaded continuous-time controllers and uncertain systems," *Circuits, Systems and Signal Processing*, vol. 17, no. 5, pp. 591–611, 1998.

[112] R. Shorten, K. S. Narendra, and O. Mason, "A result on common quadratic Lyapunov functions," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 110–113, Jan 2003.

[113] D. Srinivasagupta, H. Schättler, and B. Joseph, "Time-stamped model predictive control: an algorithm for control of processes with random delays," *Computers & Chemical Engineering*, vol. 28, no. 8, pp. 1337 – 1346, 2004.

[114] J. A. Stankovic and K. Ramamritham, “What is predictability for real-time systems?” *Real-Time Systems*, vol. 2, no. 4, pp. 247–254, 1990.

[115] S. Stuijk, M. Geilen, and T. Basten, “*SDF^3*: SDF for free,” in *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on.* IEEE, 2006, pp. 276–278.

[116] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, “Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications,” in *Embedded Computer Systems (SAMOS), 2011 International Conference on.* IEEE, 2011, pp. 404–411.

[117] Z. Sun and S. S. Ge, *Stability theory of switched dynamical systems.* Springer, 2011.

[118] K. C. Toh, M. J. Todd, and R. H. Tütüncü, “SDPT3 - A Matlab software package for semidefinite programming, Version 1.3,” *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 545–581, 1999.

[119] C. Trofino and A. De Souza, “An LMI approach to stabilization of linear discrete-time periodic systems,” *Systems & Control Letters*, vol. 45, no. 5, pp. 371–385, 2000.

[120] D. Q. Truong and K. K. Ahn, “Robust variable sampling period control for networked control systems,” *IEEE Transactions on Industrial Electronics*, vol. 62, no. 9, 2015.

[121] R. H. Tütüncü, K. C. Toh, and M. J. Todd, “Solving semidefinite-quadratic-linear programs using SDPT3,” *Mathematical programming*, vol. 95, pp. 189–217, 2003.

[122] J. Valencia, D. Goswami, and K. Goossens, “Composable platform-aware embedded control systems on a multi-core architecture,” in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 502–509.

[123] J. Valencia, E. P. van Horssen, D. Goswami, W. P. M. H. Heemels, and K. Goossens, “Resource utilization and quality-of-control trade-off for a composable platform,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 654–659.

[124] G.-J. van den Braak, C. Nugteren, B. Mesman, and H. Corporaal, “Fast Hough Transform on GPUs: Exploration of Algorithm Trade-Offs,” in *Advanced Concepts for Intelligent Vision Systems.* Springer, 2011, vol. 6915, pp. 611–622.

[125] E. P. van Horssen, S. Prakash, D. Antunes, and W. P. M. H. Heemels, "Event-driven control with deadline optimization for linear systems with stochastic delays," *IEEE Transctions on Control of Network Systems*, pp. 1–1, 2017.

[126] E. van Horssen, D. Antunes, and W. Heemels, "Switching data-processing methods in a control loop: Trade-off between delay and probability of data acquisition," *IFAC-PapersOnLine*, vol. 49, no. 22, pp. 274 – 279, 2016.

[127] P. H. Vardhana, B. K. Kumar, and M. Kumar, "A robust controller for dstatcom," in *Power Engineering, Energy and Electrical Drives, 2009. POWERENG '09. International Conference on*, 2009, pp. 546–551.

[128] A. Varga, "On solving periodic Riccati equations," *Numerical Linear Algebra with Applications*, vol. 15, no. 9, pp. 809–835, 2008.

[129] ——, "Robust and minimum norm pole assignment with periodic state feedback," *IEEE Transactions on Automatic Control*, vol. 45, no. 5, pp. 1017–1022, 1998.

[130] N. Vatanski, J.-P. Georges, C. Aubrun, E. Rondeau, and S.-L. Jämsä-Jounela, "Networked control with delay measurement and estimation," *Control Engineering Practice*, vol. 17, no. 2, pp. 231 – 244, 2009.

[131] N. Wada, K. Saito, and M. Saeki, "Model predictive control for linear parameter varying systems using parameter dependent Lyapunov function," in *Circuits and Systems, 2004. MWSCAS'04. The 2004 47th Midwest Symposium on*, vol. 3, 2004, pp. iii–133–6 vol.3.

[132] C. Wang, C.-Y. Lin, and M. Tomizuka, "Statistical learning algorithms to compensate slow visual feedback for industrial robots," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 3, 2015.

[133] Y. Wang, B. M. Nguyen, H. Fujimoto, and Y. Hori, "Multirate estimation and control of body slip angle for electric vehicles based on onboard vision system." *IEEE Trans. Industrial Electronics*, vol. 61, no. 2, pp. 1133–1143, 2014.

[134] Z. Wang, B. Huang, and H. Unbehauen, "Robust H_∞ observer design of linear state delayed systems with parametric uncertainty: The discrete-time case," *Automatica*, vol. 35, no. 6, 1999.

[135] F. Xia, Z. Wang, and Y. Sun, "Allocating iec function blocks for parallel real-time distributed control system," in *Control Applications, 2004. Proceedings of the 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 254–259.

[136] ——, "Design and evaluation of event-driven networked real-time control systems with iec function blocks," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 6. IEEE, 2004, pp. 5148–5153 vol.6.

[137] Y. Xia, G. P. Liu, P. Shi, D. Rees, and E. J. C. Thomas, "New stability and stabilization conditions for systems with time-delay," *International Journal of System Sciencie*, vol. 38, no. 1, pp. 17–24, 2007.

[138] G. Xie and L. Wang, "Controllability and stabilization of discrete-time switched linear systems," *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, vol. 6, pp. 5338–5343, 2004.

[139] L. Xie, M. Fu, and C. E. de Souza, "H $_\infty$ control and quadratic stabilization of systems with parameter uncertainty via output feedback," *IEEE Transctions on Automatic Control*, vol. 37, no. 8, pp. 1253–1256, 1992.

[140] L. Xie, "Output feedback H_∞ control of systems with parameter uncertainty," *International Journal of control*, vol. 63, no. 4, pp. 741–750, 1996.

[141] L. Xie, M. Fu, and C. E. de Souza, "H $_\infty$ control and quadratic stabilization of systems with parameter uncertainty via output feedback," *IEEE Transctions on Automatic Control*, vol. 37, no. 8, pp. 1253–1256, 1992.

[142] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 2009, pp. 30–37.

[143] M. Yu, L. Wang, T. Chu, and F. Hao, "An LMI approach to networked control systems with data packet dropout and transmission delays," in *Decision and Control, 2004 43th IEEE Conference on*, vol. 4, 2004, pp. 3545–3550.

[144] M.-Z. Yuan, Z. Wang, X. Ren, H.-B. Yu, and Y. Zhou, "Function block-based pipelined controller," in *Industrial Electronics Society, IECON 2003-Annual Conference of the IEEE*. IEEE, 2003, pp. 1956–1961.

[145] H. Yuen, J. Princen, J. Illingworth, and J. Kittler, "Comparative study of hough transform methods for circle finding," *Image and vision computing*, vol. 8, no. 1, pp. 71–77, 1990.

[146] F. Zhang, *Analysis for EDF scheduled real-time systems*. Department for Computer Science, University of York, 2009.

[147] ——, *The Schur complement and its applications*. Springer Science & Business Media, 2006, vol. 4.

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

[149] S. Zhou and J. Lam, “Robust stabilization of delayed singular systems with linear fractional parametric uncertainties,” *Circuits, Systems and Signal Processing*, vol. 22, no. 6, pp. 579–588, 2003.

[150] G. Zong, L. Hou, and H. Yang, “Further results concerning delay-dependent control for uncertain discrete-time systems with time-varying delay,” *Mathematical Problems in Engineering*, vol. 2009, 2009.

List of Symbols

Modelling and control design with pipelined sensing

$A_c \in \mathbb{R}^{n\times n}$	continuous-time state matrix 19, 47, 76, 116
$A_d \in \mathbb{R}^{n\times n}$	discrete-time state matrix 22, 47, 52, 116
$B_c \in \mathbb{R}^{n\times 1}$	continuous-time input matrix 19, 47, 76, 116
$B_d \in \mathbb{R}^{n\times 1}$	discrete-time input matrix 22, 47, 52, 116
$C_c \in \mathbb{R}^{1\times n}$	output matrix 19, 47, 76, 116
$C_d \in \mathbb{R}^{1\times (n+\gamma)}$	discrete-time augmented output matrix 23, 47, 82
$F \in \mathbb{R}$	feed-forward gain 26, 48
$\Gamma_d \in \mathbb{R}^{(n+\gamma)\times 1}$	discrete-time augmented input matrix 23, 47
$\gamma \in \mathbb{Z}^+$	number of sensing resources 21, 47, 118
$h \in \mathbb{R}^+$	sampling period 6, 20, 47
$h_{ac} \in \mathbb{R}^+$	sampling period of data-intensive sensor 22
$h_{Rt} \in \mathbb{R}^+$	sampling period based on rise time 21
$h_s \in \mathbb{R}^+$	sampling period based on serial implementation 21
$K \in \mathbb{R}^{1\times (n+\gamma)}$	feedback gain 26, 48, 58, 62
$k \in \mathbb{Z}^{\geq}$	discrete-time index 20, 22–24, 47, 79, 116
$n \in \mathbb{Z}^+$	number of continuous-time states in the plant 19, 47, 76, 116

$\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	discrete-time augmented state matrix 23, 47
$Q \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	state weight matrix of LQR 27, 48
$R \in \mathbb{R}^{1\times 1}$	input weight matrix of LQR 27, 48
$r \in \mathbb{R}$	reference 5, 26, 48, 123
$S_t \in \mathbb{R}^{+}$	settling time 26, 48, 116
$S_{t,j}(l) \in \mathbb{R}^{+}$	settling time of the particle j in the iteration l 31
$t \in \mathbb{R}^{\geq}$	time 19, 47, 76, 116
$t_k^a \in \mathbb{R}^{+}$	start time of actuation operation 20
$\tau \in \mathbb{R}^{+}$	sensing-to-actuating delay 6, 20, 47, 81
$\tau_s \in \mathbb{R}^{+}$	worst-case execution time of actuation operation 20
$\tau_c \in \mathbb{R}^{+}$	worst-case execution time of control-computation operation 20
$\tau_s \in \mathbb{R}^{+}$	worst-case execution time of sensing operation 20
$t_k^c \in \mathbb{R}^{+}$	start time of control operation 20
$t_k^s \in \mathbb{R}^{\geq}$	start time of sensing operation 20
$u(t) \in \mathbb{R}$	continuous-time input 19, 47, 76, 116
$u_k \in \mathbb{R}$	discrete-time input 23, 47, 116
$u_{k-\gamma} \in \mathbb{R}$	discrete-time controller with delayed input 22, 52
$x(t) \in \mathbb{R}^{n\times 1}$	continuous-time state vector 19, 47, 76, 116
$x_k \in \mathbb{R}^{n\times 1}$	discrete-time state vector 22, 52, 116
$y \in \mathbb{R}$	discrete-time output 47
$y_k \in \mathbb{R}$	discrete-time output 22, 23, 52, 116
$y(t) \in \mathbb{R}$	continuous-time output 19, 76, 116
$z_k \in \mathbb{R}^{(n+\gamma)\times 1}$	discrete-time augmented state vector 23, 47

PSO algorithm

$C_g \in \mathbb{R}^{+}$	global confidence 30
$C_p \in \mathbb{R}^{+}$	personal confidence 30
$\mathscr{F}(X_j(s)) \in \mathbb{R}$	fitness of the particle X_j at iteration s 30
$f \in \mathbb{Z}^{+}$	size of the design space 29

$\hat{Q}_j(l) \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	intermediate variable to compute $Q_j(l)$ 33
$\hat{R}_j(l) \in \mathbb{R}^{1\times 1}$	intermediate scalar to compute $R_j(l)$ 33
$m \in \mathbb{Z}^+$	population size 29, 30
$V_j(l) \in \mathbb{R}^{f\times 1}$	a particle's velocity 29
$w \in \mathbb{R}^+$	inertia 30
$X_j(l) \in \mathbb{R}^{f\times 1}$	a particle in position j, iteration l 29
$Xgb(l) \in \mathbb{R}$	particle with the historical best fitness in the whole swarm 29
$Xpb_j(l) \in \mathbb{R}$	particle with the historical best fitness in the position j 29

Robustness analysis

$\alpha \in \mathbb{R}^+$	positive scalar to be optimized 53, 58, 62
$\bar{\Phi}_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	discrete-time closed-loop matrix 57
$D \in \mathbb{R}^{(n+\gamma)\times n}$	constant matrix to define $\Delta\Phi_d$ 53, 58, 62
$\Delta A_c \in \mathbb{A}_c \subseteq \mathbb{R}^{n\times n}$	continuous-time uncertainty state matrix 51, 120
$\Delta A_d \in \mathbb{A}_d \subseteq \mathbb{R}^{n\times n}$	discrete-time uncertainty state matrix 52
$\Delta A_{d,wc} \in \mathbb{A}_d \subseteq \mathbb{R}^{n\times n}$	worst-case of the discrete-time uncertainty state matrix 52
$\Delta B_c \in \mathbb{B}_c \subseteq \mathbb{R}^{n\times 1}$	continuous-time uncertainty input matrix 51, 120
$\Delta B_d \in \mathbb{B}_d \subseteq \mathbb{R}^{n\times 1}$	discrete-time uncertainty input matrix 52
$\Delta B_{d,wc} \in \mathbb{B}_d \subseteq \mathbb{R}^{n\times 1}$	worst-case of the discrete-time uncertainty input matrix 52
$\Delta\Phi_d \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	uncertainty state matrix 53, 58, 62
$E \in \mathbb{R}^{n\times(n+\gamma)}$	constant matrix containing the uncertainties of $\Delta\Phi_d$ 53, 58, 62
$G \in \mathbb{R}^{n\times n}$	uncertainty matrix bounded by $G^T G \preceq I$ 53, 58, 62
$\lambda \in \mathbb{R}^+$	positive scalar defined by $\lambda = \epsilon^{-1}$ 59, 62
$O \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	positive-definite matrix defined by $O = P^{-1}$ 58, 59, 62
$\Phi_{cl} \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	discrete-time nominal closed-loop matrix 48, 57, 58, 62
$\sigma \in \mathbb{R}^+$	positive scalar defined by $\sigma = \alpha^{-2}$ 59, 62
$V_K \in \mathbb{R}^{1\times(n+\gamma)}$	free weight matrix 62
$V \in \mathbb{R}^{(n+\gamma)\times(n+\gamma)}$	free weight matrix 58, 59, 62

Reconfigurable pipelined control

$A^{MC} \in \mathbb{R}^{n\times n}$	discrete-time state matrix of the MC 81
$A^{rc} \in \mathbb{R}^{n\times n}$	discrete-time state matrix of the RC 83
$B^{MC} \in \mathbb{R}^{n\times 1}$	discrete-time input matrix of the MC 81
$B^{rc} \in \mathbb{R}^{n\times 1}$	discrete-time input matrix of the RC 83
$F^{MC} \in \mathbb{R}$	feed-forward gain of controller MC 86
$F^{rc} \in \mathbb{R}$	feed-forward gain of controller RC 86
$\gamma^{ES} \in \mathbb{Z}^{+}$	processing resources in the embedded system 76
$\Gamma^{MC} \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$	discrete-time augmented input matrix in the MC 82
$\gamma^{MC} \in \mathbb{Z}^{+}$	processing resources used in the MC 76, 79, 81
$\Gamma^{rc} \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$	discrete-time augmented input matrix in the RC 84
$\gamma^{rc} \in \mathbb{Z}^{+}$	processing resources used in the RC 79
$\gamma^{sa} \in \mathbb{Z}^{+}$	processing resources used by sporadic applications 76
$h^{MC} \in \mathbb{R}^{+}$	sampling period in MC 79
$h^{rc} \in \mathbb{R}^{+}$	sampling period in RC 79
$K^{MC} \in \mathbb{R}^{1\times(n+\gamma^{MC})}$	feedback gain of controller MC 86
$K^{rc} \in \mathbb{R}^{1\times(n+\gamma^{MC})}$	feedback gain of controller RC 86
$k_{sw} \in \mathbb{Z}^{\geq}$	discrete-time index at the switching initiation 88
$P \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$	positive-definite matrix i.e., $P \succ 0$ 96
$\Phi^{MC} \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$	discrete-time augmented state matrix for MC 82
$\Phi^{rc} \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$	discrete-time augmented state matrix for RC 84
$S \in \mathbb{R}^{(n+\gamma^{MC})\times(n+\gamma^{MC})}$	positive-definite matrix i.e., $S \succ 0$ 96
$sa \in \mathbb{Z}^{+}$	number of sporadic applications 76
$t_k^s \in \mathbb{R}^{\geq}$	start time of sensing operation at index k 79
$t_{k_{sw}}^s \in \mathbb{R}^{+}$	switching initiation 88
$u_{k-\gamma^{MC}}^{MC} \in \mathbb{R}$	discrete-time controller with delayed input for the MC 81

$u^{rc}_{k-\gamma^{rc}} \in \mathbb{R}$	discrete-time controller with delayed input for the RC 83
$x^{MC}_k \in \mathbb{R}^{n\times 1}$	discrete-time state vector of the MC 81
$x^{rc}_k \in \mathbb{R}^{n\times 1}$	discrete-time state vector of the RC 83
$y^{MC}_k \in \mathbb{R}$	discrete-time output of the MC 81
$y^{rc}_k \in \mathbb{R}$	discrete-time output of an RC 83
$z^{MC}_k \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$	discrete-time augmented state vector for MC 81
$z^{rc}_k \in \mathbb{R}^{(n+\gamma^{MC})\times 1}$	discrete-time augmented state vector for RC 83

Variable-delay control

$A_i \in \mathbb{R}^{(n+1)\times(n+1)}$	augmented integral state matrix 123
$B_i \in \mathbb{R}^{(n+1)\times 1}$	augmented integral input matrix 123
$\mathbb{D}$	set containing the representative delays of the modes 121
$h_p \in \mathbb{R}^+$	actuation period 114, 116
$\mathbb{H}$	set containing the actuation periods 121
$K_i \in \mathbb{R}$	integral gain 123
$K \in \mathbb{R}^{1\times n}$	feedback gain 123
$\mathbb{M}$	set containing the modes of the histogram 120
$Q_i \in \mathbb{R}^{(n+1)\times(n+1)}$	state weight matrix of LQR with integral action 123
$R_i \in \mathbb{R}^{1\times 1}$	input weight constant of LQR with integral action 123
$\tau_{d,k} \in \mathbb{Z}^+$	Prediction length in samples 117
$\overline{\tau}_d \in \mathbb{Z}^+$	upper bound of the prediction length in samples 118
$\overline{\tau}_c \in \mathbb{R}^+$	upper bound of the prediction length in seconds 118
$\underline{\tau}_d \in \mathbb{Z}^+$	lower bound of the prediction length in samples 118
$\hat{x}_{i,k} \in \mathbb{R}$	integral state vector 123
$\hat{x}_k \in \mathbb{R}^{n\times 1}$	discrete-time predicted state vector 116
$\hat{z}_{i,k} \in \mathbb{R}^{(n+1)\times 1}$	augmented stated for the LQR with integrator 123

List of Abbreviations

ADAS	Advanced Driver Assistance Systems
CPS	Cyber-Physical System
CQLF	Common-Quadratic Lyapunov Function
DISC	Data-Intensive Sensing Control
DTLE	Discrete-Time Lyapunov Equation
HIL	Hardware-In-the-Loop
IBC	Image-Based Control
ITA	Indexing Table Application
LMI	Linear Matrix Inequality
LQR	Linear Quadratic Regulator
MC	Maximal Configuration
MPC	Model Predictive Control
PSO	Particle Swarm Optimization
QoC	Quality of Control
RC	Reduced Configuration
RPC	Reconfigurable Pipelined Controller
SCA	Supervisory Control Application

TA	Turner Application
TDM	Time Division Multiplexing
xCPS	eXplore Cyber-Physical Systems
ZOH	Zero-Order Hold

Acknowledgments

The final outcome of this thesis comes thanks to direct and indirect collaboration with many people. Hereby I want to express some words of gratitude to them.

To start with, I want to thank my promoter and supervisor team. I could not have asked for a more balanced group of scientists to guide my learning process. First of all, I want to thank my promoter Twan Basten for the guidance during these many years and for all the support (specially in these last stages). During our meetings I learned from him how to be a good scientist without loosing the human side. Thanks to Dip Goswami for all the patience and the many, many hours that he put into guiding me. Specially thanks for always being available to solve even the most naive questions. Thanks to Sander Stuijk for not only all the technical insights but also for teaching me a bit of his pragmatism, which is reflected in his insignia question "why do you want to do that?".

I want to thank Prof. Dr. Zebo Peng (Linköping University), Prof. Dr. Ir. Gerard Smit (University of Twente), Prof. Dr. Ir. Maurice Heemels, and Prof. Dr. Kees Goossens for all the effort that they put into reading my thesis and for insightful feedback that I received from them.

I want to thank all my colleagues (and friends) from the ES group, specially Joost, Umar, Bram, João, Hadi, Juan (David), Marc (Geilen), and Amir for all the useful (and the useless too) discussions that we had during these four years. Many thanks to the scientists of FEI (Nowadays Thermo Fisher Scientific), specially to Bart Janssen, for all the technical discussions and insights that helped me to define the nature of this research. Thanks to Marja and Rian for all the support and nice conversations that we had together. I want to sincerely thank Mark Wijtvliet and Roel Jordans for hacking the code of the wireless sun blinds controller of our building. A copy of their code allowed me to keep the sun blinds up to never miss a cloud in the sky! I want to thank Joost for all the unrecognised effort that he put into creating the most plagiarized picture of xCPS (Fig. 2.1). Thanks to all my TNO colleagues for the space that they gave me to complete this thesis in the last stages. Special thanks to *the usual suspects*, who always came up with an excuse to have a beer to close a week: Erik, Jeet, and Thomas.

Thanks to all my close friends with whom I have shared many social activities (mostly eating) Andrea, Ajay, Camille, Dimitris, Garbí, Jingyue, Juan x5 (Alfaro, Camilo, Carlos, David, and Pablo), Mabel, María, Peter, Simona, Ştefana, Rolando, and Yu.

I want to thank my parents who have been always supportive in this journey. Gracias papá y mamá por todo el apoyo durante estos años. Last but not least, special thanks to Johanna for supporting me and being by my side during this process.

Róbinson Alberto Medina Sánchez
Eindhoven, June 2021

List of Publications

Publications covered in this thesis

1. R. Medina, S. Stuijk, D. Goswami, and T. Basten, “Reconfigurable pipelined sensing for image-based control,” in *Industrial Embedded Systems (SIES), 2016 11th IEEE Symposium on.* IEEE, 2016.
2. ——, “Exploring the trade-off between processing resources and settling time in image-based control through LQR tuning,” in *Proceedings of the Symposium on Applied Computing*. ACM, 2017.
3. R. Medina, J. Valencia, S. Stuijk, D. Goswami, and T. Basten, “Designing a controller with image-based pipelined sensing and additive uncertainties,” *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 3, pp. 33:1–33:26, 2019.
4. R. Medina, S. Stuijk, D. Goswami, and T. Basten, “Implementation-aware design of image-based control with on-line measurable variable-delay,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 240–245.

Publications not covered in this thesis

1. V. Dugan, R. Medina, C. M. Ionescu, and R. D. Keyser, “On the application of model-order reduction algorithms,” in *Proceedings of the International Conference on System Theory, Control and Computing (ICSTCC).* IEEE, 2012, pp. 1–6.

2. J. Hernandez, R. Medina, and M. Hernandez, “Instrumentation and design of a supervisory system for an anaerobic biodigestor,” in *Proceedings of the International Symposium on Alternative Energies and Energy Quality (SIFAE).* IEEE, 2012, pp. 1–6.

3. R. Medina, A. Hernandez, C. Ionescu, and R. D. Keyser, “Evaluation of constrained multivariable EPSAC Predictive Control Methodologies,” in *Proceedings of the European Control Conference (ECC).* IEEE, 2013, pp. 530–535.

4. S. Adyanthaya, H. Alizadeh, J. Bastos, A. Behrouzian, R. Medina, J. van Pinxten, B. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, S. Stuijk, M. Reniers, and J. Voeten, “xCPS: A Tool to eXplore Cyber Physical Systems,” in *Proceedings of the WESE: Workshop on Embedded and Cyber-Physical Systems Education.* ACM, 2015, pp. 3:1–3:8.

5. S. Adyanthaya, H. Alizadeh, J. Bastos, A. Behrouzian, R. Medina, J. van Pinxten, B. van der Sanden, U. Waqas, T. Basten, H. Corporaal, R. Frijns, M. Geilen, D. Goswami, M. Hendriks, S. Stuijk, M. Reniers, and J. Voeten, “xCPS: A Tool to Explore Cyber Physical Systems,” *SIGBED Rev.*, vol. 14, no. 1, pp. 81–95, 2017.

6. T. Basten, J. Bastos, R. Medina, B. van der Sanden, M. C. W. Geilen, D. Goswami, M. A. Reniers, S. Stuijk, and J. P. M. Voeten, “Scenarios in the design of flexible manufacturing systems,” in *System-Scenario-based Design Principles and Applications. Springer International Publishing,* 2020, Ch. 9, pp. 181–224.

7. R. Medina, S. Tabatabaei Nikkhah, D. Goswami, W. P. M. H. Heemels, S. Stuijk, and T. Basten, “Reconfigurable pipelined control systems,” *IEEE Design and Test,*, 2020, pp. 1-6.

8. R. Medina, Z. Parfant, T. Pham, and S. Wilkins, “Multi-layer predictive energy management system for battery electric vehicles,” in *IFAC world congress,* 2020, pp. 1–6.

Curriculum Vitae

Róbinson Medina was born in Ibagué, Colombia in 1988. He received his BSc in Electronics Engineering from University of Ibagué (Colombia) in 2011. During his BSc thesis, he designed and prototyped a didactic WDM-based optic fibre module for the communications laboratory. He completed his MSc in Industrial Control Engineering from Ghent University (Belgium) and University of Ibagué in 2012, as part of a dual exchange program. During his MSc thesis, he modelled a pneumatic-based energy storage system for a combine harvester. In 2013, he started to work as teaching assistant at Cundinamarca University (Colombia), where he was involved in multiple research projects related to the development of mechatronic systems for agricultural applications. In 2014, he joined the Electronic Systems group at Eindhoven University of Technology to pursue his PhD in control based on data-intensive sensing. During his PhD, he proposed techniques to design data-intensive controllers that profit from the use of embedded systems with multiprocessing capabilities. He currently works as a Scientist Innovator at TNO-automotive. His research interests are in the fields of applied control, automotive systems, and embedded systems.