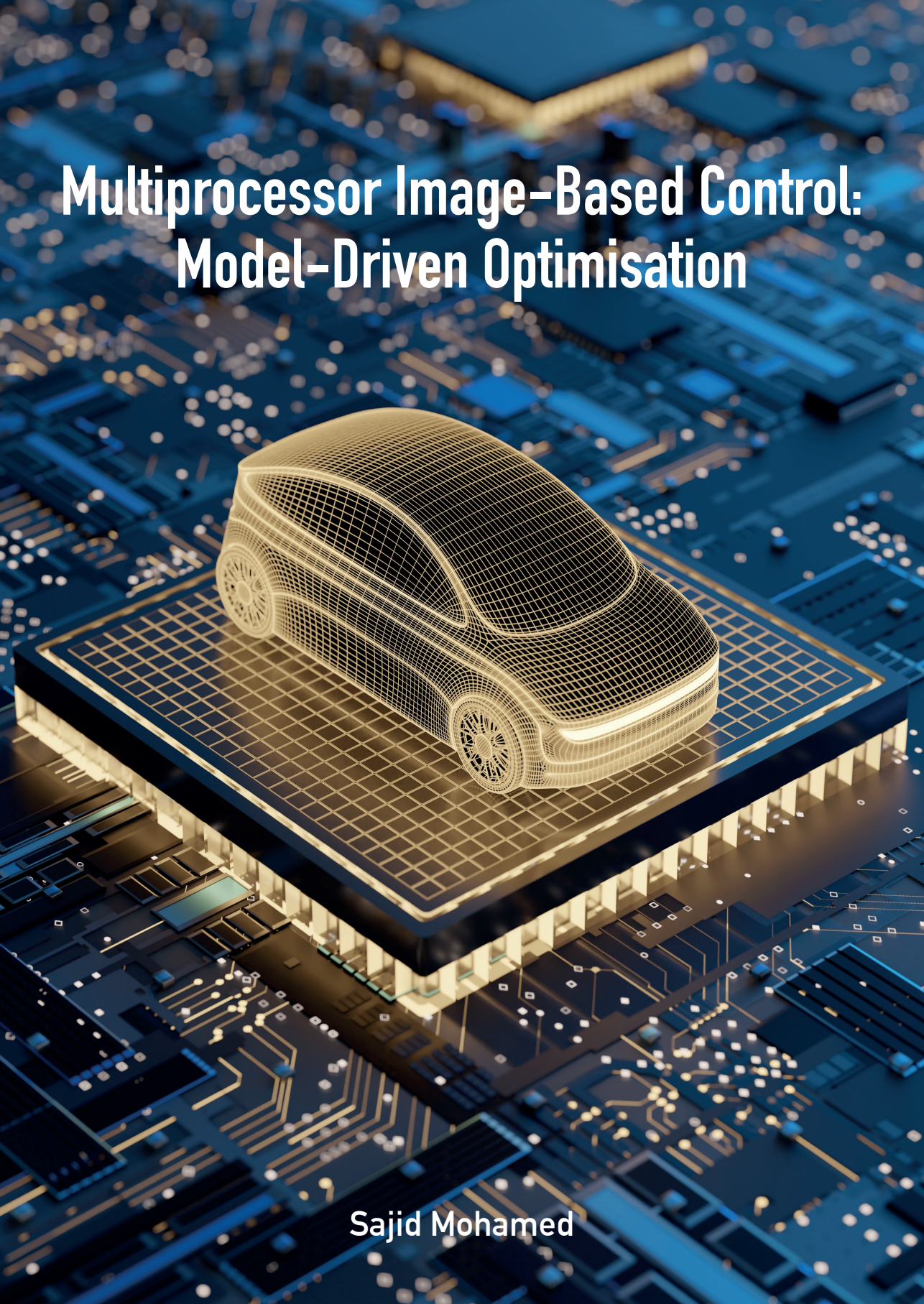
# Multiprocessor Image-Based Control: Model-Driven Optimisation



Sajid Mohamed

# Multiprocessor Image-Based Control: Model-Driven Optimisation

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 dinsdag 20 december 2022 om 16:00 uur

door

**Sajid Mohamed**

geboren te Trivandrum, Kerala, India

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

| | | |
|---|---|---|
| Voorzitter: | prof.dr.ir. M. Matters-Kammerer | |
| Promotor: | prof.dr.ir. Twan Basten | |
| Copromotor: | dr. Dip Goswami | |
| Promotie- | | |
| commissieleden: | prof.dr.ir. T. Basten | |
| | dr. D. Goswami | |
| | prof.dr. K.G.W. Goossens | |
| | prof.dr.ir. J.P.M. Voeten | |
| | prof.dr. A. Jantsch | (Technische Universität Wien) |
| | prof.dr.sc. S. Chakraborty | (UNC Chapel Hill) |

*Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.*

# Multiprocessor Image-Based Control: Model-Driven Optimisation

Sajid Mohamed

The work described in this thesis was carried out at the Eindhoven University of Technology. This work is part of the research program "oCPS" funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement No. 674875.




Typeset in LaTeX. Source: github.com/sajid-mohamed/sajidPhDThesis

Cover photo from shutterstock.
Cover design by Vincent van Zandvoord (Buro Vormvast).

Printed by Proefschriftmaken, The Netherlands.

A catalogue record is available from the
Eindhoven University of Technology Library.

ISBN 978-90-386-5580-2

Dedicated to my parents (*Umma, Vappa*) and my wife (Jemshi).

In the loving memory of my grandfathers and *vappumma*.
Your love, encouragement and prayers will always stay with me.

# Summary


## Multiprocessor Image-Based Control: Model-Driven Optimisation

Over the last years, cameras have become an integral component of modern cyber-physical systems due to their versatility, relatively low cost and multifunctionality. Camera sensors form the backbone of modern applications like advanced driver assistance systems (ADASs), visual servoing, telerobotics, autonomous systems, electron microscopes, surveillance and augmented reality. Image-based control (IBC) systems refer to a class of data-intensive feedback control systems whose feedback is provided by the camera sensor(s). IBC systems have become popular with the advent of efficient image-processing algorithms, low-cost complementary metal–oxide semiconductor (CMOS) cameras with high resolution and embedded multiprocessor computing platforms with high performance. The combination of the camera sensor(s) and image-processing algorithms can detect a rich set of features in an image. These features help to compute the states of the IBC system, such as relative position, distance, or depth, and support tracking of the object-of-interest. Modern industrial compute platforms offer high performance by allowing parallel and pipelined execution of tasks on their multiprocessors.

The challenge, however, is that the image-processing algorithms are compute-intensive and result in an inherent relatively long sensing delay. State-of-the-art design methods do not fully exploit the IBC system characteristics and advantages of the multiprocessor platforms for optimising the sensing delay. The sensing delay of an IBC system is moreover variable with a significant degree of variation between the best-case and worst-case delay due to application-specific image-processing workload variations and the impact of platform resources. A long variable sensing delay degrades system performance and stability. A tight predictable sensing delay is required to optimise the IBC system performance and to guarantee the stability of the IBC system. Analytical computation of sensing delay is often pessimistic due to image-dependent workload variations or challenging platform timing analysis. Therefore, this thesis explores techniques to cope with the long variable sensing delay by considering application-specific IBC system characteristics and exploiting the benefits of the multiprocessor platforms. Effectively han-

dling the long variable sensing delay helps to optimise IBC system performance while guaranteeing IBC system stability.

First, this thesis presents the model-driven scenario- and platform-aware design (SPADE) flow for IBC systems modelling, analysis, design and implementation. The SPADE flow expressly targets multiprocessor system-on-chip (MPSOC) platforms. The thesis develops the SPADE flow with a focus on the composable and predictable multiprocessor system-on-chip (COMPSOC) platform. The thesis further presents an adaptation of the SPADE flow for modern industrial platforms – NVIDIA Drive PX2 and NVIDIA AGX Xavier, which are closed-source and difficult to predict. The SPADE flow is validated using the performance evaluation for IMAge-based Control Systems (IMACS) framework developed as part of this thesis. The SPADE flow is explained incrementally in this thesis starting with the platform-specific aspects and later showing how the application-specific aspects are integrated.

The platform-specific aspects explored in this thesis are application parallelism and pipelining of the control loop. First, we examine the case of application parallelism with no pipelining allowed for the control loop. A scenario-aware dataflow (SADF) models the IBC system, capturing both the image-workload variations and the parallelism in the image-processing as workload scenarios for a given platform allocation. The contribution of this thesis concerning this case is the relation between dataflow timing analysis and control timing parameters to obtain a tight predictable sensing delay considering implementation constraints. Second, pipelining without parallelising the sensing application is considered. The challenge with pipelining is that the parameters which are relevant for practical implementation – inter-frame dependencies, system nonlinearities and constraints on system variables – are typically not addressed. As a contribution, this thesis presents a model-predictive control (MPC) formulation for pipelined IBC systems considering workload variations, inter-frame dependencies, system nonlinearities and constraints on system variables. Third, this thesis considers pipelining and parallelism together. It presents model transformations for modelling, analysis and mapping IBC systems using SADF. The model transformations allow to relate the dataflow timing analysis to the control timing parameters and to optimise the mapping while considering pipelining and parallelism together.

The first application-specific characteristic explored in this thesis is the impact of image-workload variations on the IBC system performance. The first contribution is the modelling and designing of IBC systems by explicitly considering image-workload variations using a scenario-based design approach. The image-workload variations are identified and modelled as a discrete-time Markov chain (DTMC), where each Markov state represents a workload scenario. At runtime, the IBC system switches between workload scenarios based on image workload. Having numerous switching workload scenarios results in an unstable system or degrades system performance. This thesis presents a controller synthesis method based on the Markovian jump linear system (MJLS) formulation. System scenarios are identified that abstract multiple workload scenarios based on camera frame

rate, sensing delay and sampling period. The DTMC is then recomputed, considering only the system scenarios, and the controller is synthesised based on the MJLS formulation. At runtime, the IBC system implementation is switching between the system scenarios. It is shown that the presented approach has a better performance compared to the state-of-the-art. This thesis also provides design guidelines on the applicability of state-of-the-art control design methods for given IBC system requirements, implementation constraints and system knowledge.

The second application-specific characteristic explored in this thesis is the impact of approximate computing on the IBC system performance. Approximate computing trades off accuracy in the signal processing for gains in response time. Approximating the camera image-signal processing (ISP) stage helps to drastically reduce the sensing delay at the cost of errors in the sensing processing. The second contribution of this thesis is the approximation-aware design of an IBC system. First, the resilience of the given IBC system to different approximation choices is analysed. Second, for each approximation choice, the sensing delay is computed, and the error due to approximations is quantified. The approximation-aware controller is then designed, for each approximation choice, by considering the sensing delay and modelling the quantified error due to approximations as sensor noise. For the analysis and validation, the thesis presents an IMACS with support for software-in-the-loop (SIL) simulation and hardware-in-the-loop (HIL) validation. The IMACS framework models the environment and dynamics using a physics simulation engine, e.g. Webots, CoppeliaSim or Matlab, and interacts with the IBC algorithm in the SIL or HIL setting.

In conclusion, this thesis aims to efficiently cope with the long variable sensing delay of IBC systems so that engineers can deploy IBC systems efficiently in time- and safety-critical domains. The thesis copes with the long variable delay by considering both application-specific characteristics and platform-specific constraints to optimise IBC system performance and stability. The proposed SPADE flow exploits the application-specific characteristics and platform-specific constraints of IBC systems to cope with the long variable sensing delay and optimise the system performance. The SPADE flow explicitly considers image-workload variations, the approximation of ISP, parallelisation of sensing processing, and pipelining of the control loop. The SPADE flow is targeted for a predictable and composable COMPSOC platform. This thesis also details how the SPADE flow can be adapted for industrial platforms. The techniques presented in this thesis achieve substantial control performance improvements compared to the state-of-the-art approaches for a given platform allocation while guaranteeing stability.

# Contents

# 1

1

# Introduction

Technology has been growing exponentially over the last few decades. The constant evolution of computers, the internet, wireless telecommunication, smartphones, cameras, and other modern technologies are drastically transforming the world we perceive and are improving our standard of living. If not for the cameras that captured my childhood days through photos and videos as memoirs, the world I grew up in would have been nothing but fiction for the current and future generations. Nowadays, cameras have become an integral part of our daily lives - be it for taking selfies using our smartphone to post on social media or for collaborating with my colleagues over a video call. Cameras, as sensors, have also become the "eyes" to perceive the surroundings for modern automotive systems, robots, autonomous systems and manufacturing systems.

Image-based control (IBC) systems that use cameras as sensors, are increasingly popular due to their low-cost and high versatility. Cameras enable the perception of depth, relative position, geometry, relative distance, colour, and tracking of the object-of-interest. As such, IBC systems are now an integral part of industrial cyber-physical systems (CPSs). CPSs refer to a class of modern industrial systems with tight interaction between computation, communication and control elements (the cyber part), and physical processes such as motion, heating/cooling, vibration, wear and tear (the physical part) within these systems. Designing CPSs requires an integrative design approach that allows for optimisation coping with the tight coordination between the cyber and the physical components [54, 72, 114]. The IBC system is compute-intensive and a standalone IBC system behaves similar to a CPS.

This thesis focuses on optimising the design and implementation of IBC systems in the resource-constrained CPS domain. The case study used throughout this thesis is a lane-keeping assist system (LKAS), though the methods are directly applicable to similar systems in the CPS domain. In this introductory chapter, first, the history of the camera and its modern significance are briefly explained. Second, the context, definition and scope of IBC systems are illustrated. Then, an overview of the state-of-the-art is given and the research challenges with the contributions of this thesis are enumerated. After providing an overview of the design and optimization flow elaborated in this thesis, finally, the motivating case study and the thesis outline are detailed.

1

## 1.1 Evolution of cameras

"A camera is an optical instrument that captures a visual image. The camera body has a small hole (the aperture) that allows light through to capture an image on a light-sensitive surface (usually photographic film or a digital sensor). Lenses focus the light entering the camera, and the size of the aperture can be widened or narrowed. A shutter mechanism determines the amount of time the photosensitive surface is exposed to light" [23]. The invention of the camera has been traced back to the work of Alhazen (Abū ʿAlī al-Ḥasan ibn al-Ḥasan ibn al-Haytham). Alhazen invented the pinhole camera *camera obscura*, and explained its scientific principles in his magnum opus *Book of Optics* (*Kītāb al-Manāẓir*) in the 11$^{th}$ century AD [3]. The camera obscura was originally used for viewing solar eclipses instead of looking directly at the sun and damaging the eye.

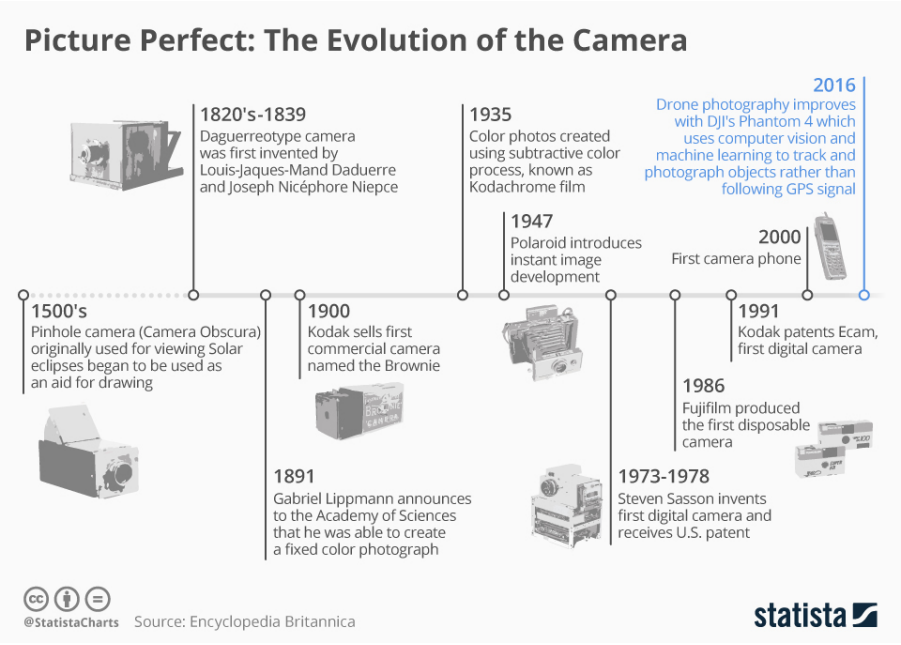



Figure 1.1: The evolution of the camera [45].

The camera has evolved over the last millennium in technology, functionality and versatility. A brief overview of the evolution of the camera (that uses visible light) is illustrated in Fig. 1.1. Artists used the first pinhole cameras to draw the outline of their paintings from real landscapes. Then, the cameras were used to preserve memories and store the image information in a physical format. The landmark moment was the invention of the digital camera, where the image is recreated from the light falling on a charge-coupled device (CCD) instead of a photo-

graphic film. The first digital camera was invented by Llyod and Sasson in 1975 and patented in 1978 [80]. However, at that time, it did not gain the necessary recognition it deserved [43] due mostly to managerial decisions. One of the reasons for non-acceptance was that it took 23 seconds to record a captured image into a cassette tape (for storage). The first professional digital single-lens reflex (SLR) camera was created by Sasson and Hills in 1989 and patented in 1991 [116]. "It had a 1.2 megapixel sensor, and used image compression and memory cards" [43].

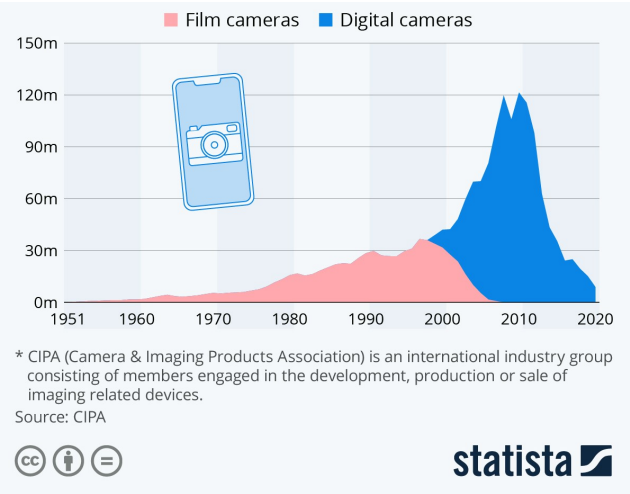



Figure 1.2: Worldwide shipments of photo cameras [109].

The invention of the professional digital camera revolutionised the camera market and also increased the global camera market size and revenue. The growth of photo camera sales is illustrated in Fig. 1.2. Two interesting points to note are: (i) the domination of digital cameras over film cameras that wiped out the normal use of film cameras; and (ii) the drastic decline in sales of digital photo cameras. The latter is due to the advancements in technology and the integration of high-quality cameras in smartphones and tablets. Buying a modern smartphone with a high-quality camera is generally preferred over a single-purpose photo camera.

Advancements in low-cost complementary metal–oxide semiconductor (CMOS) image-sensor technology [39] and Moore's law [95] enabled the faster integration of cameras in smartphones and modern industrial systems. Moore's law predicted that the number of transistors in integrated circuits would double every two years, and this has been happening over the last many decades. This enabled the miniaturisation of the size of the semiconductor component. Alternatively, many more CMOS sensors, processors and memories can be densely packed into a semiconductor component of the same size. This reduced the overall cost and size of the camera, thus enabling the widespread use of cameras in smartphones, laptops, security devices, video surveillance, drones, autonomous systems, mod-

ern industrial systems, and so on. For instance, the latest Samsung Galaxy S21 FE 5G smartphone has four cameras - a 12MP ultra-wide camera, a 12MP wide-angle camera, an 8MP telephoto camera, and a 32MP selfie camera [50].

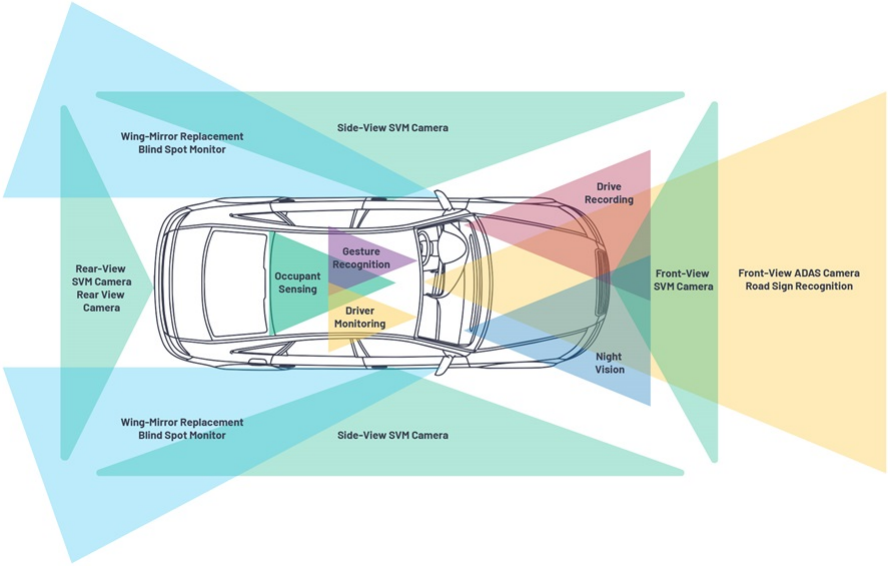



Figure 1.3: Proliferation of cameras in modern vehicles [133].

The latest Tesla autopilot system has eight cameras [128], which are necessary for the three-dimensional perception of the surrounding environment [41]. In the automotive domain, there is an immense proliferation of cameras in modern vehicles (illustrated in Fig. 1.3). According to an industrial report from Yole [141], the number of cameras in a single vehicle is expected to grow even further. Yole estimates 11 cameras per vehicle by 2024 for functionalities like surround-view, advanced driver assistance systems (ADASs), night vision, e-mirror replacement and driver monitoring. Yole predicts that cameras would also be an integral part of fully autonomous systems. Additionally, different types of cameras (based on the wavelength of light) are gaining significance for varied purposes, such as hyperspectral cameras [11], thermal-imaging cameras [76] (using infrared light) and laser imaging (used in lidar sensors [78]). Cameras have thus proven to be irreplaceable for modern applications and systems in the coming decades.

## 1.2 Image-based control systems

Cameras are now an integral part of modern (industrial) systems and are becoming increasingly popular in mixed-criticality systems. A mixed-criticality system is

a system that can execute several applications of different criticality levels - safety-critical, mission-critical and low-critical [22]. The criticality levels are formally defined, for example, as safety integrity levels in the IEC 61508 standard [12] and automotive safety integrity levels (ASILs) in the automotive ISO 26262 standard [60]. The versatility of the camera sensor allows an image captured by a single camera sensor to be used for multiple mixed-criticality applications. In the automotive domain, for example, cameras are used to perceive the surrounding environment, enabling safety-critical autonomous driving and use for non-critical applications like drive recording.

IBC systems are a class of data-intensive feedback control systems whose feedback is provided by image-based sensing using cameras as sensors. Data-intensive feedback control systems are common nowadays due to advancements in CPSs [135]. IBC systems have become popular with the advent of efficient image processing algorithms and low-cost CMOS cameras with high resolution [29]. The combination of the camera and the image-processing algorithm gives necessary information on parameters such as relative position, geometry, relative distance, depth perception and tracking of the object-of-interest. This enables the effective use of low-cost camera sensors to enable new functionality or replace expensive sensors in cost-sensitive industries like automotive [29, 101, 110]. Applications of IBC are found in robotics [29], autonomous vehicles [40, 101], advanced driver assistance systems (ADAS) [16], electron microscopes [44], visual navigation [25] and so on.

The popularity of modern IBC systems can be attributed to

- the impact of Moore's law [95] and the constant breakthroughs in semiconductor manufacturing technology. This enabled the availability of powerful multiprocessors at relatively low cost and size, and also paved the way for the miniaturisation of CMOS cameras.

- the availability of low-cost high-resolution CMOS cameras with good quality and small size.

- the versatility of the camera images and the breakthroughs in artificial intelligence (AI) technology and deep-learning algorithms [71]. Using deep-learning methods and algorithms, a multitude of features can be efficiently extracted from camera images and used for a variety of mixed-criticality applications.

- the industrial adoption of heterogeneous platforms and the breakthroughs in graphical processing units (GPUs) and neural processing units (NPUs). Examples of such developments are Tesla's full self-driving (FSD) computer [126] and NVIDIA's Drive AGX platform [97]. The FSD computer is based on a system-on-chip (SoC) that integrates industry standard components such as central processing units (CPUs), image-signal processing (ISP), GPUs, and NPUs.

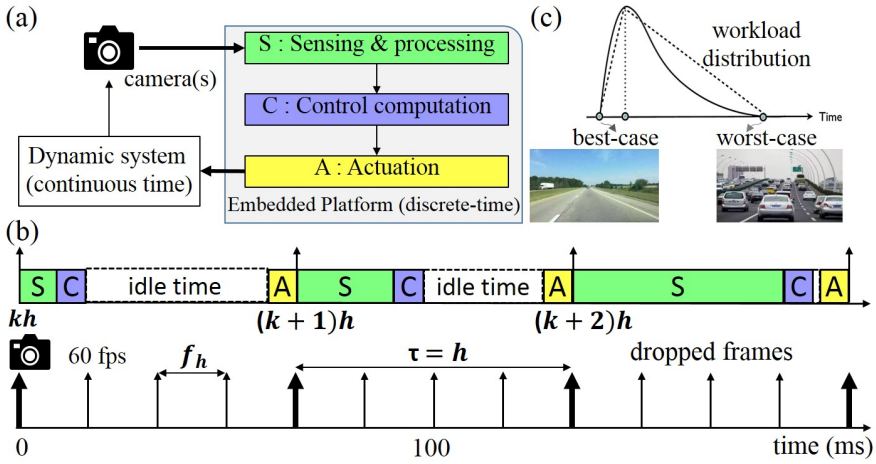



Figure 1.4: An image-based control (IBC) system: (a) block diagram; (b) Gantt chart for a typical IBC implementation; (c) workload variations captured as a distribution. In the context of the thesis, workload refers to the image workload (unless specified differently). Image workload refers to the number of features in the image that should be processed. For example, more features in an image typically implies a higher workload.

In this thesis, the focus is on IBC systems that are functionally critical and on feedback control systems whose state is measured by the image-based sensing and processing, i.e., state feedback. The case study of an automotive LKAS system is used to explain the concepts and results of this thesis. A typical IBC system is illustrated in Fig. 1.4 (a). A camera captures image frames at a pre-defined constant frames per second *fps*, i.e., the frame rate, from the dynamic system environment (with the camera frame-arrival period $f_h = \frac{1}{fps}$). An ISP processes the RAW camera image frames in the Bayer domain and converts it to the standard RGB image format, e.g. JPEG. Then, a compute-intensive image-processing algorithm processes the image frames to detect features in the image such as objects, traffic signs and lanes. These features are then used to compute the states of the system, such as relative position and distance [29]. A controller computes the control input for actuation (e.g., change in direction) using the computed states. The actuation task applies the computed control input to the IBC system.

A typical feedback control implementation sequentially and periodically (with sampling period $h$) executes the sensing and processing task (S), control compute task (C) and the actuating task (A) (as illustrated in Fig. 1.4 (b)). In an IBC system, the sensing task may have a long, variable execution time and incur a long sensing delay. Variability in execution time may occur due to variation in image-processing workload and/or in the platform load caused by other applications. The key *challenge* is to deal with this *high dynamic computation demand while guaranteeing*

*performance and meeting safety requirements* such as stability. A long variable processing delay results in dropping some camera frames from processing.

These variations can be captured statistically using a probability distribution [1] (illustrated in Fig. 1.4 (c)). It is interesting to note that this delay distribution is platform-dependent. Platform-related parameters - for instance, number of processing cores, processor clock speed, memory, and communication bandwidth - have a direct impact on the observed delay. A long worst-case sensing delay leads to a long sensor-to-actuator delay $\tau$ (the time between the start of a sensing task and the end of the corresponding actuation task) and thus results in degraded control performance [8, 117].

In this thesis, we develop approaches to cope with a long variable sensing delay exploiting the benefits of a multiprocessor platform and the application-specific IBC system characteristics. The platform-aware aspects explored in this thesis that exploit the benefits of a multiprocessor platform are - application parallelisation and pipelining of the control loop. Parallelisation refers to executing sensing subtasks in parallel and thereby reducing the delay compared to the sequential implementation. Pipelining refers to the pipelined execution of the control loop over multiple processing cores thereby reducing the effective sampling period (the time between the start of two successive sensing tasks). The two application-specific characteristics we exploit are - the image workload variations and approximate computing. Image workload variations occur due to the variations in the number of features in the captured images and the platform load. Approximate computing trades off accuracy in the signal processing for gains in response time and energy.

## 1.3 Implementation platforms

A platform refers to the combination of hardware resources (computation, communication, and memory) and the software (tasks, messages, mapping and scheduling) required to deploy an algorithm/application on the hardware effectively. The execution time of tasks - sensing and processing (S), control compute (C), and actuation (A) - of an IBC system is dependent on the platform configuration. The relatively straightforward way to reduce the long sensing delay is to have a processor with a high clock speed. However, this increases the cost of the processor. Moreover, the scaling of clock speed has reached its limits because of physical constraints in chip manufacturing and operation. Another option to cope with long sensing delay is to have a platform with multiprocessor capabilities that enable concurrent parallel and pipelined execution of tasks. A multiprocessor platform is economical due to the impact of Moore's law and is common nowadays.

In this thesis, two kinds of multiprocessor platforms are considered for the IBC system implementation. First, a composable and predictable multiprocessor system-on-chip (CompSOC) platform [55] is considered. The CompSOC platform offers a composable and (timing) predictable implementation for the designer. The results of this thesis can be effectively applied to such predictable platforms

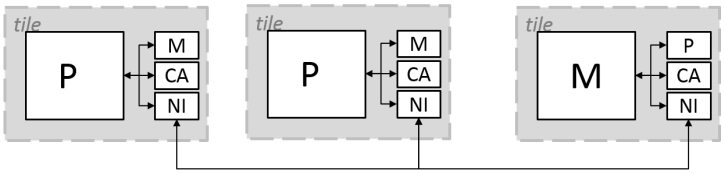



Figure 1.5: A CompSOC platform with two processor tiles and a memory tile connected through a network-on-chip (NoC).

without any adaptation. Second, state-of-the-art industrial platforms are considered. Predictability and composability are usually not offered in an industrial platform. We adapt our design approach for the industrial platforms NVIDIA Drive PX2 and NVIDIA AGX Xavier [47] to demonstrate its applicability in an industrial context. Our design approach is also applicable for more recent industrial platforms like the Tesla FSD computer [126] and NVIDIA Drive AGX [97].

### 1.3.1 CompSOC

CompSOC offers a tile-based architecture [122] (see Fig. 1.5). Each tile has a processor $P$, memory $M$, communication assist $CA$ and network interface $NI$. Each processor tile has a microblaze processor, the memory tile contains an external memory interface, e.g., dynamic random access memory (DRAM), and the NoC provides interconnection between the tiles. The platform is predictable with tight bounds on worst-case execution times (WCETs) of tasks and composable so that applications sharing the platform do not interfere with each other. A scheduler performs (re)configuration and time-triggered task execution.

### 1.3.2 NVIDIA Drive PX2

The NVIDIA Drive PX2 [97] platform consists of two Tegra SoCs that communicate to each other via ethernet. Each Tegra SoC has two CPU clusters (see Fig. 1.6). One cluster contains four ARM Cortex A57 cores and the other contains two NVIDIA Denver2 cores. The clusters are connected through a high-performance network interconnect. Each of the Tegra SoCs also has two GPUs - an integrated Pascal GPU (iGPU) and a discrete GPU (dGPU) with maximum clock rates of 1.27 and 1.29 GHz respectively. The iGPU has 256 CUDA cores and the dGPU has 1154 CUDA cores. The GPUs are accessed via the respective CPUs in the SoC. The Ubuntu 16.04 LTS operating system (OS) runs on the CPU platform.

### 1.3.3 NVIDIA AGX Xavier

The NVIDIA AGX Xavier platform (illustrated in Fig. 1.7) consists of a Xavier SoC and other components explained in [47]. The CPU complex consists of four het-

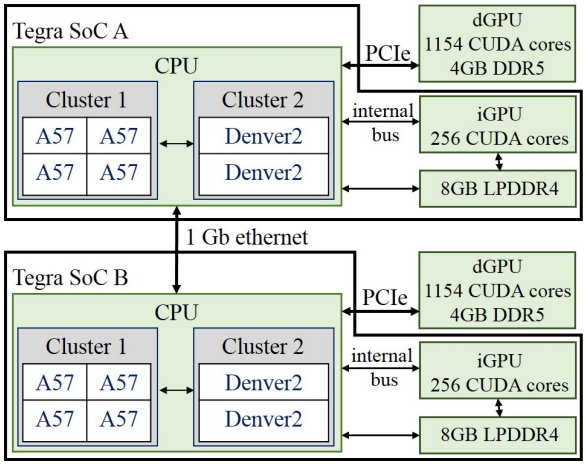



Figure 1.6: NVIDIA Drive PX2 platform structure. LPDDR4 and DDR5 are the memory blocks. Each CPU cluster also has internal instruction and data memory (not shown in the graph).

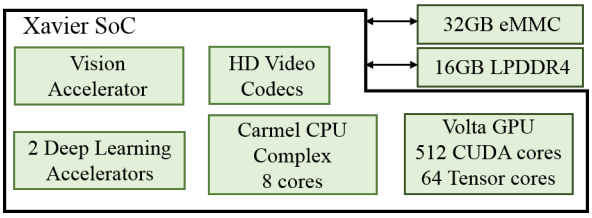



Figure 1.7: NVIDIA AGX Xavier platform block diagram. LPDDR4 and eMMC are the memory blocks. Each CPU cluster also has internal instruction and data memory (not shown in the graph).

erogeneous dual-core NVIDIA Carmel CPU clusters based on ARMv8.2 with a maximum clock frequency of 2.26GHz. The GPU with a maximum clock frequency of 1.37GHz is accessed via the CPUs in the SoC. The Ubuntu 18.04 LTS OS runs on the CPU platform.

## 1.4 State-of-the-art

This thesis explores techniques to cope with the long variable sensing delay by considering application-specific IBC system characteristics and exploiting the benefits of multiprocessor platforms. Effectively handling the long variable sensing delay helps to optimise IBC system performance while guaranteeing IBC system stability. State-of-the-art methods do not fully exploit the IBC system character-

istics and advantages of the multiprocessor platforms for optimising the sensing delay. The IBC system sensing delay is also variable with a significant degree of variation between the best-case and worst-case delay due to application-specific image-processing workload variations and platform characteristics. A long variable sensing delay degrades system performance and stability. A tight predictable sensing delay is required to optimise the IBC system performance and to guarantee the stability of the IBC system. Analytical computation of sensing delay is often pessimistic due to image-dependent workload variations or challenging platform timing analysis.

Relevant literature deals with the following questions:

- What are the relevant overall design approaches for such embedded control problems?
- What are the relevant IBC system modelling and analysis techniques?
- What are the techniques for IBC system design to deal with long delays in a feedback control loop?

**Design approaches:** An IBC system is often designed based on the *separation-of-concerns* principle between the control theory and embedded systems disciplines [7, 110]. Co-design of control and scheduling is another design paradigm explored in the literature [24]. The emphasis is on platform-based design methods that take into account platform resource constraints while designing the controller [7, 53]. Contract-based design [113] is a platform-based design paradigm for cyber-physical systems where the interactions between control theory and embedded design are defined based on contracts.

From the embedded-systems discipline, a system-scenario-based design approach [51] is proposed where different behaviours (scenarios) of an application are explicitly considered to avoid over-dimensioning or sub-optimal performance due to worst-case design. Identifying, characterising and modelling these scenarios and dealing with the runtime scenario transitions are specific for each application and generally not trivial. In this thesis, we combine scenario-based and platform-based design methods into a coherent scenario- and platform-aware design (SPADE) approach for IBC system design and implementation.

**IBC system modelling:** Model-based design [74, 130] approaches focus on designing applications based on abstract models of application and platform such that the implementation is guaranteed to behave with predictable performance. Numerous model-of-computation (MOC) are available in literature [26, 31, 124, 129]. The approaches in this thesis do not depend on a specific MOC. They need a MOC that can capture the dynamic behaviours (scenarios) of the application, can analyse timing and has support for platform-aware mapping analysis.

**IBC system design - control engineering perspective:** The main challenge in designing an IBC system is to cope with the long sensing delay. Control engineers

tackle a long delay using advanced state estimation [136], robust design [65], predictive control [81], observer-based [135] methods, and multi-rate sampling [49] methods. These methods rely heavily on the system model and are vulnerable to modelling errors with longer delays. Also, control engineers typically design an IBC system considering the worst-case image workload [110] and do not explicitly consider the platform constraints. In literature, workload variations are typically considered only for sequential IBC implementation [46] and not for parallel and pipelined implementations [112], [69]. In [82], pipelining is considered along with variable delay. However, these approaches do not consider system nonlinearities, i.e., the variations in system dynamics, and constraints imposed on the system variables, which can be crucial when considering a practical implementation.

**IBC system design - embedded systems perspective:** Embedded systems engineers aim to reduce processing delay and period by parallel implementations of the algorithms using heterogeneous multiprocessor platforms having specialised hardware such as GPUs [2] and FPGAs [66]. Pipelined control [69, 112] is another approach targeting homogeneous multiprocessor implementations that reduce the effective sampling period without changing the processing delay. Inter-frame dependencies, i.e. the data or algorithmic dependencies between consecutive frame processing, e.g. due to video coding [77] or visual tracking [120], and resource sharing between pipes, which are crucial aspects for a practical pipelined IBC system implementation, are not explicitly considered in the literature. Additionally, we observe that an integral approach to pipelined parallelism can provide immense benefits to the IBC system design and implementation. However, such an integral approach is lacking in literature.

**IBC system design - approximate computing perspective:** Approximate computing is another technique that can help reduce the processing delay by trading off computation accuracy for speed. Approximations reduce the compute workload across algorithm, architecture and circuit levels. Commonly used algorithmic approximations are computation skipping [83], precision scaling [131] and replacing error-resilient compute-intensive functions with neural networks [42]. A learning approach to design ISPs for new camera systems is proposed in [62]. At the architecture level, research efforts have focused on both approximating general-purpose processors [27] as well as domain-specific accelerators [33]. At the circuit level, research efforts focus on manual design techniques for adders and multipliers [63], as well as automated methodologies for designing energy-efficient approximate circuits [34].

Approximation benefits in a camera-based biometric security system, using an iris recognition application, is showcased in [57]. An approximate smart camera system is introduced in [104], using camera resolution scaling, reducing memory refresh rate and computation skipping. An approximate ISP pipeline tuned for computer-vision algorithms is designed in [21], by skipping selected ISP stages. An algorithm-hardware co-designed system is showcased in [145]. However, these approximation techniques do not consider a closed-loop feedback system. Ap-

proximation decisions in a closed-loop system have quality implications at a later point in time. Optimising a system while accounting for the temporal approximate behaviour is not explored in any of the mentioned work.

## 1.5 Research challenges and contributions

The primary research question addressed in this thesis is:
**How to cope with the long variable sensing delay in an IBC system?**

This thesis deals with challenges to cope with the long variable sensing delay in an IBC system and the scientific contributions are broadly classified into two directions – exploitation of the benefits of the multiprocessor platforms through a platform-aware design and exploitation of the application-specific IBC system characteristics.

In this context, along the first line of research, we exploit the benefits of the multiprocessor platforms through application parallelisation and pipelining of the control loop. The approaches proposed in this thesis maximise the quality-of-control (QoC) of the controller implementation for a given multiprocessor platform allocation. In the following, we outline the three scientific contributions related to parallelisation, pipelining and pipelined parallelism.

**Contribution 1 (SPADE flow for IBC system design considering parallelisation):** Parallelisation refers to executing sensing subtasks in parallel and thereby reducing the delay compared to the sequential implementation considered before. A parallel implementation is possible when multiple cores are allocated for the sensing algorithm. It is, however, limited by the degree of parallelism of the sensing algorithms. The state-of-the-art literature related to parallel implementation for an IBC system does not consider workload variations which is crucial for maintaining optimal closed-loop performance or QoC, as already explained. Moreover, a parallel implementation in an IBC system is highly influenced by platform-specific parameters such as camera frame-rate, number of available cores and so on. At the end, these design aspects lead to the two important control design parameters – sampling period and delay. Starting with a given, often pessimistic, sampling period and delay, like what can be noticed in the state-of-the-art [69], the overall design often leads to a low QoC.

In this thesis, first, we introduce the SPADE approach for parallel implementation. The SPADE approach uses a formal model-of-computation – dataflow – for the entire IBC system including sensor processing. This enables us to model workload variation as well as platform-specific design parameters that are crucial for design optimality of a IBC system. The contribution of this thesis concerning this case is the relation between dataflow timing analysis and control timing parameters to obtain a tight predictable sensing delay considering implementation constraints. The image workload variations are captured in workload scenarios and modelled using a scenario-aware dataflow (SADF). Then, workload scenarios

are mapped to the given platform allocation considering the platform constraints such as the camera frame rate and the number of available cores. Mapping an SADF results in a binding-aware graph.

In this thesis, we relate the latency and throughput of the binding-aware graph to the delay and sampling period of the controller design. Our approach enables a scenario- and platform-aware design and optimisation of the IBC system. Our approach exploits frequently occurring workload scenarios to optimise control performance. During controller design, we then optimize performance and guarantee stability by identifying appropriate system scenarios and by designing a switched controller that switches between those scenarios. The initial proposed approach assumes that the platform is predictable, i.e., the WCET of the sensing algorithm for the same image input is bounded and predictable over multiple executions on the platform. However, the execution times cannot be bounded for industrial platforms. In this thesis, we also show how we can adapt our SPADE approach for parallelisation for an industrial platform where we propose to use frequently occurring task execution times instead of WCET.

The first contribution has been published in:

- Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario-and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 75:103037, 2020.
- Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. Optimising quality-of-control for data-intensive multiprocessor image-based control systems considering workload variations. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 320–327, 2018.

With respect to the state of the art, this thesis proposes the first formal model-based design framework for the IBC system co-design that relates dataflow analysis and controller design. The SPADE approach brings together dataflow and control formalisms in the same framework. This relation allows to bring in the optimisation techniques from the dataflow domain into the control timing parameter optimisation. The proposed SPADE approach also makes a step forward towards real-life implementation by detailing how the flow can be adapted for industrial platforms. Both academic and industrial platforms could implement the SPADE approach. software-in-the-loop (SIL) and hardware-in-the-loop (HIL) simulation results validate our claim.

**Contribution 2 (Extending the SPADE flow considering pipelining for IBC system design):** Pipelining refers to the pipelined execution of the control loop over multiple processing cores. It is an alternative to exploit the availability of multiple cores on the platform to address the challenge of long delay in an IBC system. In an IBC system, a pipelined implementation implies the start of processing a new camera frame while one or more frames are still being processed. In essence, this increases the processing throughput of the frames, which reduces the closed-loop sampling period $h$ (the time between the start of two successive sensing tasks).

Although such implementation does not reduce the sensing delay, a shorter sampling period opens the door for achieving a higher QoC with an appropriate controller design.

Given that multiple frames are processed simultaneously, a crucial aspect are the inter-frame dependencies that some of the sensing algorithms might impose. Inter-frame dependencies refer to the data or algorithmic dependencies between consecutive frame processing, e.g., due to video coding [77] or visual tracking [120]. Moreover, for a real-life implementation, it is crucial to consider system nonlinearities and constraints on system variables. System nonlinearities refer to the variations in system dynamics, and constraints need to be imposed on the system variables. For example, the maximum steering angle of the Udacity self-driving car is set to +/- 25 degree [132] and the vehicle velocity is kept constant for the simulations in [68]. While pipelined implementations for the IBC system is considered in the state-of-the-art as potential direction [112], [69], the existing approaches do not consider several key practical aspects needed for their real-life adoption.

As a contribution, this thesis presents an extended SPADE approach with an adaptive predictive control formulation based on linear parameter-varying (LPV) input/output (I/O) models for a pipelined multiprocessor implementation of IBC systems while considering workload variations, inter-frame dependencies, system nonlinearities and constraints, and thus makes a step forward towards real-life adoption. The inter-frame dependencies are characterised using the platform constraints - camera frame rate and the number of available cores, and the control timing parameters - delay and sampling period.

The second contribution has been published in:

- Sajid Mohamed, Nilay Saraf, Daniele Bernardini, Dip Goswami, Twan Basten, and Alberto Bemporad. Adaptive predictive control for pipelined multiprocessor image-based control systems considering workload variations. In *59th IEEE Conference on Decision and Control (CDC)*, 2020.

The state-of-the-art pipelined multiprocessor IBC implementation approaches do not consider inter-frame dependencies, system nonlinearities and constraints on system variables. This thesis proposes a model-predictive control (MPC) formulation that consider these aspects and makes a step forward towards real-life adoption. The simulation results and discussions validate our claim.

**Contribution 3 (Extending the SPADE flow considering pipelined parallelism for IBC system design[1]):** In an IBC system, a long variable sensing delay results in a long sampling period and a long delay from the controller viewpoint. To deal with them, modern multiprocessor IBC implementations consider either parallelisation of the sensing task or pipelining of the control loop. In a pipelined implementation, the sampling period is shortened while the delay remains long. Such implementation does not require knowledge of the internal processing structure

[1]The SPADE flow controller design is open sourced and can be accessed on github: https://github.com/sajid-mohamed/SPADeControlDesign

of the sensing workload. The benefit of such methods is restricted by the platform parameters such as number of available cores and camera frame-rate. On the other hand, in a parallelised implementation, the delay is shortened allowing for a shorter sampling period. This line of approaches require computational insight (i.e., model-of-computation) of the sensing task for a parallelised implementation. Similar to the pipelined case, the benefits of parallelization methods are restricted by the number of available cores and the degree of parallelism present in the sensing algorithms. The impact of both parallelisation and pipelining together on the QoC of IBC systems is not explored in the literature while it is rather obvious direction given their complimentary nature.

This thesis proposes a model-based design method for multiprocessor IBC implementation, considering both parallelisation and pipelining together. It presents model transformations for modelling, analysis and mapping IBC systems using SADF. The model transformations allow to relate the dataflow timing analysis to the control timing parameters and to optimise the mapping while considering pipelining and parallelism together. In this thesis, the particular problem we address is: For a given platform allocation, what is the optimal degree of pipelining and degree of parallelisation required to maximise the QoC? A unified SPADE algorithm is proposed that takes into account image-workload variations, inter-frame dependencies and platform constraints. The application is efficiently modelled and analysed using a scenario-aware dataflow graph, and an implementation-aware switched controller is designed that optimises QoC and guarantees stability. Two types of platforms are considered - predictable, e.g., CompSOC, and industrial platforms, e.g., NVIDIA AGX Xavier. The SPADE approach performs the best when the platform is predictable and the execution times are bounded. In an industrial platform where the execution times cannot be bounded, an adaptation of the SPADE approach is also proposed.

The third contribution has been published in:

- Sajid Mohamed, Dip Goswami, Sayandip De, and Twan Basten. Optimising multiprocessor image-based control through pipelining and parallelism. *IEEE Access*, 9:112332–112358, 2021.

The state-of-the-art multiprocessor IBC implementation approaches do not consider pipelined parallelism explicitly. This thesis details the pipelined parallelism implementation along with the required model transformations for realizing SPADE using the SADF MoC. This thesis also details the adaptation of the SPADE approach for industrial platforms for pipelined parallelism. SiL and HiL simulation results validate our claim.

Further, as already explained, this thesis exploits the opportunities of two application-specific characteristics - image-workload variations and approximate computing. In the following, we outline the specific scientific contributions along these two directions.

**Contribution 4 (IBC system design considering workload variation):** For sens-

ing, an IBC system needs to process image workload, which is data-intensive and experiences significant variation. Image-workload variations occur due to the variations in: i) the number of features in the captured images, and ii) the platform load. Typically, an IBC system considers the worst-case image workload [110] and the impact of these variations are not explicitly considered in the controller design. Moreover, the platform constraints such as the number of available cores and camera frame-rate are not considered in the state-of-the-art design approaches. Since such approaches are built upon the worst-case characteristics, they often lead to low overall QoC of the system for a given platform allocation.

In our proposed approach in this thesis, we profile all the workload relevant for the IBC system, identify the frequently occurring workload scenarios, and model the workload variations using a discrete-time Markov chain (DTMC). A DTMC characterises the frequently occurring workload scenarios and the probability of transition between these different scenarios. For a given workload scenario, we further perform platform-aware optimization including binding and mapping to multiple cores as well as optimized controller design. When the number of the workload scenarios becomes high, it often become challenging to ensure stability and high QoC. Then, system scenarios that abstract multiple workload scenarios are identified based on platform constraints. Finally, a Markovian jump linear system (MJLS) controller is designed for the system scenarios. We demonstrate significant improvement in terms of QoC using the proposed method over the state-of-the-art worst-case-based designs. With respect to the SPADE flow, the MJLS controller is an alternative controller design method that explicitly models workload variation.

The fourth contribution has been published in:

- Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *58th Conference on Decision and Control (CDC)*, pages 3997–4004. IEEE, 2019.

In literature, variable sensor-to-actuator delay (resulting from the variable workload) was dealt with through switched linear controllers with either known or unknown sequences of delay occurrences. These solutions either suffer from poor performance (from unknown arbitrary delay sequences) or unrealistic assumptions (when knowledge of the exact delay sequence is not available in reality). This thesis proposes an alternative controller design method based on the MJLS formulation that models the delay occurrences as a DTMC. The simulation results and discussions show that the proposed approach performs better when sufficient system knowledge is available.

**Contribution 5 (Approximation-aware IBC system design):** Approximate computing trades off accuracy in the signal processing for gains in response time and energy. In this thesis, the focus is on how we use approximate computing for gains in response time which is further exploited in the IBC system design. The gains in energy, reported in [36], is excluded in this thesis as it is not the key goal in this the-

sis. The camera ISP pipeline transforms a RAW image in the Bayer domain to pixels in the RGB domain through a series of image enhancing stages. Approximating the camera ISP pipeline helps to drastically reduce the sensing delay at the cost of errors in the sensing processing. There are various classes of approximation – fine grained and coarse grained. In an IBC system, a few of these compute-intensive image enhancing stages could potentially be skipped/approximated without compromising the IBC system functionality. Generally, usage of approximate computing for ISP is an interesting direction given its potential to reduce the sensing delay (which is a key challenge for designing an IBC system). However, approximate computing for the feedback control system is an unknown territory in literature. There are two key challenges – (i) how to model the effect of approximation in the IBC system and guarantee stability and QoC in presence of approximation in the feedback signal processing, (ii) how to test and evaluate the impact of such a design paradigm since it is an important step to identify when and how much approximation is suitable for a given design problem.

In this thesis, we first propose an integrated performance evaluation framework that combines environments, controller design and computation platform. The performance evaluation for IMAge-based Control Systems (IMACS) framework[2] helps to test and evaluate the impact of approximation in closed-loop feedback systems. First, the resilience of the given IBC system to different approximation choices is analysed. Second, for each approximation choice, the sensing delay is computed, and the error due to approximations is quantified using the IMACS framework. Then, an approximation-aware controller is designed, for each approximation choice, by considering the sensing delay and modelling the quantified error due to approximations as sensor noise. The approximation-aware design offers alternatives in terms of sensing (with approximate ISP) and controller (i.e., approximation-aware controller) in SPADE flow.

The IMACS framework is published in:

- Sajid Mohamed, Sayandip De, Konstantinos Bimpisidis, Vishak Nathan, Dip Goswami, Henk Corporaal, and Twan Basten. IMACS: A Framework for Performance Evaluation of Image Approximation in a Closed-loop System. In *8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2019.

In literature, a performance evaluation framework that can analyse the impact of injecting errors in the image processing on the closed-loop IBC system performance was not present. IMACS is the first open-source framework that enables the performance evaluation of image approximation in closed-loop IBC systems. SIL and HIL simulation results validate our claim.

The approximation-aware controller design is published in:

[2]The IMACS framework is open sourced and can be accessed on github: https://github.com/sajid-mohamed/imacs

- Sayandip De, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 8: 174568–174586, 2020.

- Sayandip De, Sajid Mohamed, Konstantinos Bimpisidis, Dip Goswami, Twan Basten, and Henk Corporaal. Approximation trade offs in an image-based control system. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1680–1685. IEEE, 2020.

In literature, image approximation and platform constraints are not explicitly considered in the design of the controller of the IBC system. This thesis proposes an approximation-aware control design that takes as input the quantified error due to approximation. SIL and HIL simulation results validate our claim.

## 1.6 SPADE flow overview

This thesis presents a scenario- and platform-aware design flow (SPADE) for IBC systems that brings together the essence of key scientific contributions made in this thesis (explained in Section 1.5). An overview of our SPADE approach is illustrated in Fig. 1.8, summarised below and explained in detail in subsequent chapters.

1. Formal modelling of the IBC system: An IBC application is captured as an IBC SADF considering workload variations $W$ and the platform as a platform graph. Further, an *implementation-aware* IBC SADF captures the given design parameters - camera frame arrival period $f_h$, maximum number of allowed pipes $p$, total number of available cores $n_c^{avl}$ and allocated processing cores for parallel execution per pipe $n_c^{//}$. The design parameters fully determine the implementation choice - non-pipelined without parallelism, non-pipelined with parallelism, pipelined without parallelism and pipelined with parallelism. The parallelism here refers to the parallel execution of sensing subtasks limited by the degree of parallelism of the IBC application.

2. Analysis and design: We map the implementation-aware IBC graph for each workload $s_i \in W$ to the platform graph to obtain the *binding-aware graph* $\mathscr{G}_i^b$ for that specific workload using the SDF3 mapping flow [122]. $\mathscr{G}_i^b$ is a synchronous dataflow graph (SDFG) that models the mapping of the implementation-aware graph to the platform graph. The mapping binds each actor in the SDFG to a processing core in the platform graph. For the ordering of execution of actors bound to the same core, a static-order schedule is encoded in the SDFG. A throughput and latency analysis of $\mathscr{G}_i^b$ yields the sensor-to-actuator delay $\tau_i$, and sampling period $h_i$. For a pipelined implementation, the throughput analysis of the worst-case image-workload

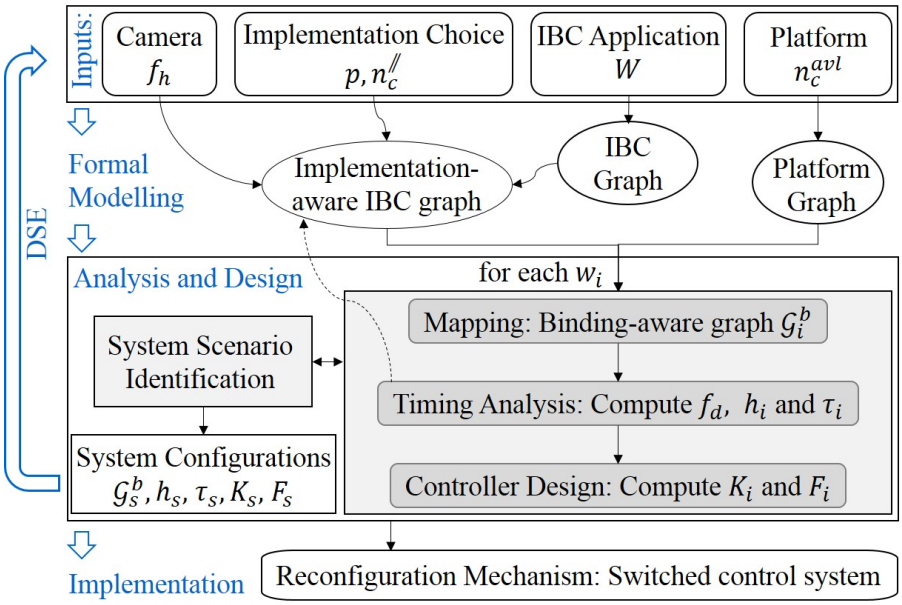



Figure 1.8: Overview of our SPADE design flow for pipelined parallelism. $W$ is the set of varying workloads and $w_i$, $\mathcal{G}_i^b$, $\tau_i$, $h_i, K_i$ and $F_i$ are the workload, binding-aware graph, sensor-to-actuator delay, sampling period, feedback gain and feedforward gain for a workload scenario $s_i$ (determined by $w_i \in W$); $\mathcal{G}_s^b$, $\tau_s$, $h_s, K_s$ and $F_s$ are the corresponding parameters for an identified system scenario $s_s$ (that abstract multiple workload scenarios). $f_h$ is the camera frame arrival period, $f_d$ is the inter-frame dependence time, $p$ is the number of pipes for pipelining, $n_c^{//}$ is the number of cores allocated for parallelism per pipe, and $n_c^{avl}$ is the total number of available cores. For the scope of the thesis, the pipelined implementation is always periodic.

scenario allows to compute the inter-frame dependence time $f_d$ (explained later in Section 4.5.4). If $f_d > h_i$, the implementation-aware graph is updated with the realisable period and $\tau_i$ and $h_i$ are recomputed. The controllers are then designed for the resulting ($\tau_i$, $h_i$) to obtain the controller feedback and feedforward gains ($K_i$, $F_i$). Trying to cater to the designed workload scenarios at runtime means that we have a switching system. A switching system with too many switching states is challenging for controller stability and may result in poor performance. Hence, we aggregate multiple workload scenarios with similar control timing parameters as a *system scenario*. A system scenario $s_s$ abstracts multiple workload scenarios and has a constant ($\tau_s$, $h_s$) during implementation. A *system configuration* is de-

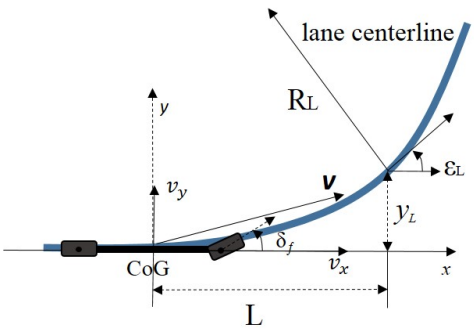
fined as the combination of mapping and controller configurations, i.e. $\mathcal{G}_s^b$, $\tau_s$, $h_s$, $K_s$, and $F_s$ (as explained later in Section 4.3.2). It should be noted that the controller gains (i.e., $K_s$, and $F_s$) can be designed using different approaches depending on the design goals. In this thesis, the SPADE flow is illustrated using the linear quadratic regulator (LQR), linear-quadratic-integral (LQI), linear-quadratic-Gaussian (LQG), MPC, and MJLS controller design approaches. Typically, the number of identified system scenarios is low in number, and the idea is that switching between the system scenarios at runtime guarantees stability and improved performance. For pipelined parallelism, a design-space exploration (DSE) using the SPADE flow needs to be performed by varying the design parameters to identify the best implementation choice (parameters $p, n_c^{//}$, further explained in Section 4.6.1).

3. Runtime implementation: The system configurations for the implementation choice are stored in a look-up table (LUT) in platform memory for the runtime implementation. Dynamic runtime reconfiguration may be needed since there can be a switching behaviour between system configurations due to image-workload variations.

## 1.7 Motivating case study: LKAS

This thesis uses the motivating case study of a LKAS. The bicycle model derived from [68] (illustrated in Fig. 1.9) is considered for simulating the LKAS of a vehicle[3]

[3]The (default) vehicle parameters are those specified in [68] for Honda Accord.




$v_x$: vehicle longitudinal velocity
$v_y$ : vehicle lateral velocity
$R_L$: curve lane radii
$L$ : look-ahead distance at centre of gravity of vehicle (CoG)
$y_L$: lateral deviation at $L$
$\varepsilon_L$: angle between the tangent to lane centerline and vehicle orientation
$\delta_f$: front tire steering angle

Figure 1.9: LKAS dynamics model derived from [68].

on a straight road and it is described as follows,

$$A_c(v_x) = \begin{bmatrix} -\frac{c_f+c_r}{mv_x} & \frac{-mv_x^2+c_r l_r-c_f l_f}{mv_x} & 0 & 0 \\ \frac{-l_f c_f+l_r c_r}{I_\psi v_x} & -\frac{l_f^2 c_f+l_r^2 c_r}{I_\psi v_x} & 0 & 0 \\ -1 & -L & 0 & v_x \\ 0 & -1 & 0 & 0 \end{bmatrix},$$

$$B_c = \begin{bmatrix} \frac{c_f}{m} & \frac{l_f c_f}{I_\psi} & 0 & 0 \end{bmatrix}^\top,$$

$$C_c = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix},$$

$$D_c = \mathbf{0},$$

where, referring to Fig. 1.9, we define the state vector $x(t) = [v_y, \dot{\psi}, y_L, \varepsilon_L]$, the measured output $y(t)$ as $y_L$, and the control input $u(t)$ as the steering angle $\delta_f$, where $\dot{\psi}$ is the vehicle's yaw rate in rad/s, where the velocity components $v_x$ and $v_y$ are in m/s, where $l_f$, $l_r$ (= 1.22 and 1.62 m respectively) denote distance of the front and rear axles from the centre of gravity (CoG), where $I_\psi$ (= 2920 kg·m$^2$) is the total inertia of the vehicle around its CoG, where $c_f$, $c_r$ (= $1.2 \times 10^5$ N/rad) denote cornering stiffness of the front and rear tires, and where the total mass of the vehicle is $m$ (= 1590 kg). Note that the model can be either linear time-invariant or time-variant depending on whether longitudinal velocity $v_x$ is constant or time-varying.

## 1.8 Thesis outline

This thesis is organized along the research challenges and scientific contributions outlined in Section 1.5. Accordingly, the five scientific contributions are reported in Chapter 2 to Chapter 6. Chapters 2, 3 and 4 focus on platform-aware design aspects where the research focus has been on reducing sampling period and delay by parallel implementation (Chapter 2), pipelined implementation (Chapter 3) or their combination (Chapter 4). Chapters 5 and 6 report design approaches exploiting application-specific characteristics. Chapter 5 reports the proposed approach on dealing with workload variation in a sequential IBC implementation. Chapter 6 presents the approximation-aware design of an IBC system. It presents the IMACS framework for approximate computing in closed-loop systems. Chapter 7 makes some concluding remarks and outlines a number of relevant future extensions.

# 2

# SPADE by parallelisation

2

Image-based control (IBC) systems are increasingly being used in various domains including autonomous driving. The key challenge in IBC is to deal with high computation demand while guaranteeing performance and safety requirements such as stability. While modern industrial heterogeneous platforms, such as NVIDIA Drive, offer the necessary compute power, application development on these platforms with performance and safety guarantees is still challenging. Alternative time-predictable platforms are not yet in widespread use.

A typical design flow for IBC systems consists of three distinct elements: (i) *mapping* tasks onto platform resources; (ii) *timing analysis*, consisting of *task-level* worst-case execution time (WCET) analysis and *application-level* analysis to obtain worst-case performance bounds on aspects such as latency and throughput; (iii) *controller design* using the obtained performance bounds, ensuring performance and safety. While such a three-step design process is modular in nature, it usually leads to over-dimensioned systems with sub-optimal performance, because task- and/or application-level timing bounds are pessimistic.

We present a basic scenario- and platform-aware design flow for IBC systems that exploits *frequently occurring workload scenarios* to optimize performance. For industrial platforms, where tight task-level WCET bounds are difficult to obtain, we moreover propose to use *frequently occurring task execution times* instead of WCET estimates to obtain tight application-level temporal bounds. During controller design, we then optimize performance and guarantee stability by identifying appropriate *system scenarios* and by designing a *switched controller* that switches between those scenarios. We illustrate our method considering a predictable multiprocessor system-on-chip platform - the composable and predictable multiprocessor system-on-chip (COMPSOC). We validate the proposed method using hardware-in-the-loop (HIL) experiments with an industrial heterogeneous multiprocessor platform - NVIDIA Drive PX2 - considering a lane-keeping

The content of this chapter is an adaptation of the following two papers:
Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario-and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 75:103037, 2020.
Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. Optimising quality-of-control for data-intensive multiprocessor image-based control systems considering workload variations. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 320–327, 2018.

assist system (LKAS). Both platforms and the LKAS case study were already introduced in Chapter 1. We obtain an improved control performance compared to state-of-the-art IBC design.



## 2.1 Background and contributions

IBC systems are a class of data-intensive feedback control systems whose feedback is provided by image-based sensing using a camera. Data-intensive feedback control systems are common nowadays due to advancements in cyber-physical systems (CPSs) [135]. IBC systems have become popular with the advent of efficient image processing algorithms and low-cost complementary metal–oxide semiconductor (CMOS) cameras with high resolution [29]. The combination of the camera and the image processing algorithm gives necessary information on parameters such as relative position, geometry, relative distance, depth perception and tracking of the object-of-interest. Applications of IBC are found in robotics [29], autonomous vehicles [40, 101], advanced driver assistance systems (ADAS) [16], electron microscopes [44], visual navigation [25] and so on.

As illustrated in Fig. 2.1, a classical control implementation sequentially and periodically executes the sensing task, control compute task and actuating task. In an IBC system, the sensing task has a long, variable execution time, incurring a long delay. Variability in execution time may occur due to variation in image-processing workload and/or in the platform load caused by other applications. The key challenge is to deal with this high dynamic computation demand while guaranteeing performance and meeting safety requirements such as stability.

IBC applications are nowadays usually implemented on some heterogeneous multiprocessor platform that may be shared with other applications. A typical design flow for IBC applications is composed of three distinct elements: (i) *mapping* tasks onto platform resources, which may be done manually or (semi-)automatically; (ii) *timing analysis*, consisting of *task-level* execution-time analysis and *application-level* analysis to obtain worst-case application-level latency and throughput bounds; (iii) *controller design* ensuring performance and safety guarantees taking into account task- and application-level temporal bounds. A typical flow abstracts variable task execution times through WCET estimates. These are often overly conservative, because of image-dependent workload variations and/or

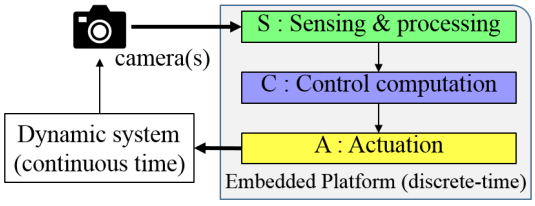



Figure 2.1: An IBC system: block diagram (repeating Fig. 1.4 (a), for readability)

difficult to predict platform timing. This leads in turn to loose application-level timing bounds which hampers controller design. The resulting IBC system has sub-optimal control performance and is often over-dimensioned.

Fig. 2.2 illustrates a standard IBC implementation on a single processing core. A camera captures image frames with a period $f_h$, the inverse of which is referred to as the frame rate. The frame rate determines the number of image frames that arrive per time unit, e.g., frames per second (FPS). Typically, the camera frame rate is much higher than the rate at which the frames can be processed on a single core. Pessimistic task-level WCET estimates and application-level analysis result in over-allocation of processing resources for the worst-case workload and idling for non-worst-case workloads (to keep the sampling period constant). This leads to a long sampling period $h$. In the example of Fig. 2.2, with one processing core, we can only process every third image frame, resulting in sub-optimal control performance.

In this chapter, we present the basic version of the model-based SPADE for multiprocessor IBC systems, already outlined in Section 1.6, that exploits *parallelization of the sensing task* and *frequently occurring workload scenarios* to optimize performance. For industrial platforms, where tight task-level WCET bounds are difficult to obtain, we moreover propose to use *frequently occurring task execution times* instead of WCET estimates to obtain tight, though possibly no longer conservative, application-level temporal bounds for workload scenarios. For controller design, we identify appropriate *system scenarios* [51] that take into account platform mapping and controller performance for specific workload scenarios. Each system scenario corresponds to a specific sampling period. IBC performance

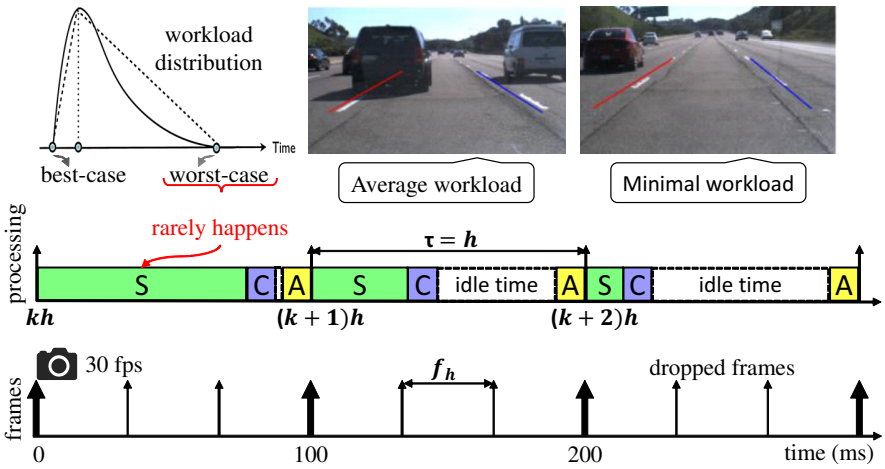



Figure 2.2: Illustration of classical IBC system implementation considering worst-case scenario. S: sensing and image processing, C: control computation and A: actuation. (Adapted from Fig. 1.4 (b) and (c), for readability.)

is then optimized and stability is ensured by designing a *switched controller* that switches at runtime between system scenarios. Scenario-aware dataflow (SADF) [129] is used as a model of computation to capture parallelized (workload and system) scenarios and for timing analysis.



Predictable platforms, such as COMPSOC [55] and PRET [38], provide predictable tight WCETs for individual tasks in an application. Further, the composability property of such platforms ensures that other applications sharing the platform do not interfere with the application under consideration. The WCET variations of a task execution on predictable platforms is mainly due to the image workload variations. These properties make predictable multiprocessor system-on-chip (MPSOC) platforms suitable for model-based design. We develop and illustrate our SPADE approach for the predictable and composable interference-free COMPSOC platform.

We further apply our method on the industrial NVIDIA Drive PX2 platform. Industrial heterogeneous platforms provide high compute power with support for extensive parallelization that is typically needed for data-intensive applications. However, such platforms are closed-source and use operating systems (OSs) that result in high variations in execution times of application tasks mapped to the platform. An ensuing challenge is that the application timing is difficult to predict. Derived task-level WCET estimates are overly pessimistic. Model-based approaches using these pessimistic WCETs lead to pessimistic application-level performance bounds. This potentially compromises control performance and may lead to resource over-provisioning. However, task execution time distributions due to workload and platform-dependent variations can be statistically analysed from observed data, e.g., as a PERT distribution [1] (illustrated in Fig. 2.2). Such a distribution allows to classify the most frequently occurring task execution times. Using those execution times give tighter, though possibly no longer conservative application-level performance bounds. SPADE copes with possible timing analysis violations in the (switched) controller design. Using SPADE, we perform model-based design-space exploration (DSE) for an industrial setup over resource utilisation, quality of control and energy consumption to obtain Pareto-optimal system configurations at design time. We consider the concrete case study of a LKAS implemented on the NVIDIA Drive PX2 platform, sharing the platform with two other data-intensive applications - object detection and tracking and automatic emergency braking.

**Contributions:** In this chapter,

1. we present the basic SPADE flow for IBC system design considering parallel implementation.

2. we compare SPADE with a state-of-the-art pipelined control approach [112] through simulations for a predictable MPSOC platform - COMPSOC. Pipelined control does not parallelize the sensing but uses multiple cores to pipeline multiple sensing instances. We provide a guideline when the SPADE approach is suitable with respect to the pipelined control approach.

3. we adapt SPADE targeting an industrial platform - NVIDIA Drive PX2. We show that we can leverage the principles of predictable model-based (co-)design for industrial platforms by carefully co-designing the image-processing implementation and the switched controller design, using a system-scenario-based approach [51].

4. we validate the SPADE approach in an industrial setting using a HIL experiment.



## 2.2 Embedded image-based control

We consider a setting for an IBC system as shown in Fig. 2.1. Our sensor is the camera module that captures the image stream. The image stream is then fed to an embedded multiprocessor platform at a fixed frame rate (FPS). The tasks in our application - compute-intensive image sensing and processing (S), control computation (C) and actuation (A) - are then mapped to run on this multiprocessor.

### 2.2.1 Linear time-invariant (LTI) feedback control systems

We consider an LTI feedback control system given by:

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

where $x(t) \in \mathbb{R}^n$ represents the *state,* $y(t) \in \mathbb{R}$ represents the *output* to be regulated and $u(t) \in \mathbb{R}$ represents the control *input* of the system at any time $t \in \mathbb{R}_{\geq 0}$. $A_c$, $B_c$ and $C_c$ represent the system, input and output matrices of appropriate dimensions.

We illustrate our work using the motivating case study of a vision-based lateral control system model explained in Section 1.7. The tasks in our LKAS - compute-intensive image sensing and processing (S), control computation (C), and actuation (A) - need to be mapped to run onto a multiprocessor platform. Quality-of-control (QOC) needs to be optimized. The state-space matrices of our LKAS for the vehicle speed of 15m/s are as follows,

$$A_c = \begin{bmatrix} -10.06 & -12.99 & 0 & 0 & 0 \\ 1.096 & -11.27 & 0 & 0 & 0 \\ -1.000 & -15.00 & 0 & 15 & 0 \\ 0 & -1.000 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B_c = \begin{bmatrix} 75.47 \\ 50.14 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; C_c = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

A fifth state is added for observability [68] of our controller compared to the equations in Section 1.7. The fifth state is the curvature of the road at the look-ahead distance, and helps to observe the road curvature. The control input $u(t)$ is the

front wheel steering angle $\delta_f$ and the output $y(t)$ is the lateral deviation at the look-ahead distance $y_L$.

### 2.2.2 Embedded implementation



Implementation of an IBC system involves the execution of three sequential tasks: *sensing and processing* ($S$), *control computation* ($C$) and *actuation* ($A$). These tasks repeat; let the start and finish times of the task instances be given by $t_s(.)$ and $t_f(.)$, respectively. The execution times of $S^k$, $C^k$ and $A^k$ (the $k$-th instance) are then given by,

$$e_T^k = t_f(T^k) - t_s(T^k),$$

where $T \in \{S, C, A\}$. The interval between two consecutive executions of sensing tasks $S^k$ and $S^{k+1}$ is then the *sampling period* $h^k$ for the $k$-th instance. The time interval between the starting time of $S^k$ and finishing time of $A^k$ is the *sensor-to-actuator delay* $\tau^k$ for the $k$-th instance.

$$h^k = t_s(S^{k+1}) - t_s(S^k), \qquad \tau^k = t_f(A^k) - t_s(S^k).$$

We consider a time-triggered implementation of tasks $S$, $C$ and $A$. A camera captures the images at discrete intervals, e.g., 30 fps, and the image frame arrival period $f_h$ is given by,

$$f_h = \frac{1}{\text{frame rate}}; \text{ for 30 fps, } f_h = \frac{1}{30}s = 33.33 \text{ ms}.$$

This means that the sampling period $h$ needs to be an integer multiple of $f_h$. Sensor processing is followed by control computation and actuation operations which generally take a short and nearly constant time for execution. A sensing operation takes much longer time, i.e.,

$$e_S \gg e_C + e_A$$

where $e_S$, $e_C$ and $e_A$ are the worst-case execution times of sensing and processing, control computation and actuation, introduced above. Moreover, $t_s(C^k) = t_s(S^k) + e_S$ and $t_s(A^k) = t_s(C^k) + e_C$. The total (worst-case) execution time of the control loop is then given by $e_{\text{total}} = e_S + e_C + e_A$.

The effective sensor-to-actuator delay $\tau$ and sampling period $h$ are then given by,

$$\tau = e_{\text{total}}, \; h = \lceil \frac{e_{\text{total}}}{f_h} \rceil f_h.$$

We assume that the start of sensor data processing is aligned with the camera frame arrival and the actuation is delayed to guarantee constant sensor-to-actuator delay. With sensor-to-actuator delay $\tau$ and a zero-order-hold mechanism with sampling period $h \in \mathbb{R}$, $u(t)$ becomes piecewise constant in the intervals $t \in [kh + \tau, \; (k+1)h + \tau]$ for $k \in \mathbb{Z}_{\geq 0}$.

Image-processing workloads may vary, e.g., depending on image content. In Fig. 2.2, for example, the number of features in an image determines the workload. Each workload scenario $s_i$ is annotated with a pair $(h_i, \tau_i)$ that models the sampling period and delay associated with it. A zero-order sample-and-hold approach can then be used to discretize the system based on the workload scenario $s_i$. Eq. (2.1) can be reformulated as follows:

$$\begin{aligned} x[k+1] &= A_{s_i} x[k] + B_{0,s_i} u[k] + B_{1,s_i} u[k-1], \\ y[k] &= C_c x[k] \end{aligned} \tag{2.2}$$

where,

$$\begin{aligned} A_{s_i} &= e^{A_c h_i}, \\ B_{0,s_i} &= \int_0^{h_i-\tau_i} e^{A_c s} ds \cdot B_c,\ B_{1,s_i} = \int_{h_i-\tau_i}^{h_i} e^{A_c s} ds \cdot B_c \end{aligned} \tag{2.3}$$

In Eq. (2.2), we assume that $u[-1] = 0$ for $k = 0$. We define new system states $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$ with $z[0] = \begin{bmatrix} x[0] & 0 \end{bmatrix}^T$ to obtain a higher-order augmented system as follows to obtain a delay-free state space:

$$z[k+1] = A_{aug,s_i} z[k] + B_{aug,s_i} u[k],\ y[k] = C_{aug} z[k] \tag{2.4}$$

where,

$$A_{aug,s_i} = \begin{bmatrix} A_{s_i} & B_{1,s_i} \\ 0 & 0 \end{bmatrix},\ B_{aug,s_i} = \begin{bmatrix} B_{0,s_i} \\ I \end{bmatrix},\ C_{aug} = \begin{bmatrix} C_c & 0 \end{bmatrix}.$$

0 and $I$ represent the zero and identity matrices of appropriate dimensions. A check for controllability [37] is done for this augmented system. If the system is not controllable, controllability decomposition is done to obtain a controllable subsystem. We can apply standard control design techniques for the delay-free state-space model shown in Eq. (2.4).

### 2.2.3 Control law and control configurations

In view of the augmented system of Eq. (2.4), we use a *state feedback* controller $u[k]$ of the following form,

$$u[k] = K_{s_i} z[k] + F_{s_i} r_{ref}$$

where $K_{s_i}$ is the state feedback gain and $F_{s_i}$ is the feedforward gain both designed for the workload scenario $s_i$. $r_{ref}$ is the reference value for the controller. The design of gains can be done with state-of-the-art control design techniques such as linear quadratic regulator (LQR) or pole-placement [37]. Note that any other state-of-the-art control design technique can also be used for designing these gains.

For each workload scenario $s_i$, we then define a *control configuration* $\chi^c_{s_i}$ as a tuple $\chi^c_{s_i} = (h_i, \tau_i, K_{s_i}, F_{s_i})$.

### 2.2.4 Controller stability

At runtime, the workload scenarios are switching based on the image workload variations and/or platform load. This switching behaviour can lead to system instability. Therefore, we must *guarantee* stability of the overall system while improving QoC.



**Theorem 2.2.1.** (Stability criterion [125]) *Consider $A_{aug,s_i}$ to be discrete-time* LTI *systems. $V(z) = z^T Pz$ is the common quadratic Lyapunov function (*CQLF*) of the systems $A_{aug,s_i}$ if there exist $P = P^T > 0$, $Q = Q^T > 0$ and $P$ is the simultaneous solution of the discrete-time Lyapunov equations,*

$$A_{aug,s_i}^T P A_{aug,s_i} - P = -Q < 0. \tag{2.5}$$

*The existence of a* CQLF *is a sufficient condition for the stability of a system with switching subsystems.*

We transform the stability condition (Eq. (2.5)) into linear matrix inequalities (LMIs) to analyse for the existence of a CQLF. The analysis equation, Eq. (2.6), is obtained by performing the following operations: i) substitute $A_{aug,s_i}$ in Eq. (2.5) with $A_{aug,s_i} = A_{aug,s_i} + B_{aug,s_i} K_{s_i}$, ii) apply Schur complement, and iii) left- and right- multiplication by diag$(P^{-1}, I)$ and set $Q = P^{-1}$.

$$\begin{bmatrix} -Q & QA_*^T + QK_{s_i}^T B_*^T \\ A_*Q + B_* K_{s_i} Q & -Q \end{bmatrix} < 0,\ Q > 0 \tag{2.6}$$

where $A_* = A_{aug,s_i}$, $B_* = B_{aug,s_i}$ for each scenario $s_i$. If a solution exists, then the switching subsystems are stable. The choice of scenarios need to be modifies if a solution does not exist. A less aggressive mode with poorer performance is usually more likely to meet the stability condition. Failure to guarantee switching stability would result in a classical worst-case based design.

An alternate controller synthesis method is proposed for this setting using a Markovian jump linear system (MJLS) formulation in Chapter 5.

### 2.2.5 Control performance: mean square error (MSE)

The mean square error (MSE) is the mean of the cumulative sum of the squared errors, i.e.:

$$MSE \quad = \quad \frac{1}{n} \sum_{k=1}^{n} (y[k] - r_{ref})^2$$

where $n$ is the number of observations, $y[k]$ is the value of the $k^{th}$ observation and $r_{ref}$ is the reference value. The MSE quantifies, in essence, how fast the output $y(t)$ reaches the reference $r_{ref}$. A lower MSE implies a better QoC.

## 2.3 Model-of-computation (MoC) for IBC

Our application is modelled using a MoC or a programming model that allows timing analysis. Our model should capture dynamic behaviour and scenario-awareness. This enables us to model and analyse execution time variations that happen at runtime due to either image workload variations and/or platform load. We assume that WCET estimates of task workloads are given for a platform or can be computed.



A MoC is required to compute the parameters relevant for the control design - the sampling period $h$ and the sensor-to-actuator delay $\tau$. However, the challenge now is: How to accurately determine $h$ and $\tau$ at design time for a multiprocessor, possibly heterogeneous, platform implementation? The choice of binding and scheduling of tasks on the platform determines $h$ and $\tau$.

### 2.3.1 Scenario-aware dataflow (SADF)

We choose SADF [129] as the formal MoC for our application as it enables us to: i) model dynamic behaviour, analyse timing, and optimally map application tasks to the platform for maximising the effective utilisation of allocated resources, ii) relate throughput of the dataflow graph to the sampling period, and thus combine dataflow analysis and mapping with control design parameters and QoC, and iii) to efficiently implement a runtime mechanism that manages necessary dynamic reconfiguration between system scenarios.

Following the formalisation of [4], an SADF model is a tuple $(\Sigma, \mathscr{F})$, where

- $\Sigma = \{s_i \mid s_i = (w_i, \mathscr{G}_i),\ w_i \in W\}$ is a set of scenarios being a set of pairs of workloads $w_i$ and their corresponding synchronous dataflow graphs (SDFGs) $\mathscr{G}_i$;
- the ($\omega$-)language $\mathscr{F}$ describes a set of infinite scenario sequences represented using $\omega$-regular expressions of scenarios $s_i \in \Sigma$.

Here, the workload refers to the image workload, i.e., the number of features in the image that should be processed. For example, more features in an image imply a higher workload. We assume that workloads are totally ordered, i.e., for any two workloads $w_i$ and $w_j$, either $w_i \leq w_j$ or $w_j \leq w_i$. Ordering the workloads helps to prune the state-space for system-scenario identification (explained later in Section 2.5.3).

An SDFG [73] is a tuple $\mathscr{G} = (\mathscr{A}, \mathscr{C}, e, r_p, r_c, i)$ where $\mathscr{A}$ is a finite set of actors, $\mathscr{C} \subseteq \mathscr{A}^2$ the set of channels, $e : \mathscr{A} \rightarrow \mathbb{R}_{\geq 0}$ returns for each actor its associated firing delay. The firing delay models the time it takes to execute (fire) an actor. If an actor models a computational task, the firing delay typically models the (worst-case) execution or response time of that task. $r_p : \mathscr{C} \rightarrow \mathbb{N}_{>0}$ is a function that returns for each channel its production rate, $r_c : \mathscr{C} \rightarrow \mathbb{N}_{>0}$ is a function that returns for each channel its consumption rate, $i : \mathscr{C} \rightarrow \mathbb{N}_0$ returns for each channel its number

of initial tokens. When actors of an SDFG fire, they consume and produce tokens according to the specified consumption and production rates. They can only fire if sufficient tokens are available on their input channels.

A *repetition vector* $\rho$ of an SDFG $\mathcal{G}$ is a function $\rho : \mathcal{A} \longrightarrow \mathbb{N}_0$ such that for every channel $c = (a_m, a_n) \in \mathcal{C}$, $r_p(c) \times \rho(a_m) = r_c(c) \times \rho(a_n)$. A repetition vector $\rho$ for an SDFG $\mathcal{G}$ is called non-trivial iff for all $a_m \in \mathcal{A}$, $\rho(a_m) > 0$. An SDFG is called consistent iff it has a non-trivial repetition vector. For a consistent SDFG, the unique smallest non-trivial repetition vector is designated as the repetition vector $\rho$ of the SDFG. An SDFG *iteration* is a minimal non-empty set of actor firings that has no net effect on the token distribution in the graph. For a consistent SDFG, for all $a_m \in \mathcal{A}$, the set contains $\rho(a_m)$ firings of $a_m$. For the scope of this work, we assume that the IBC application model, which is an SADF, can only have consistent SDFGs and the SDFGs are deadlock-free. These assumptions can be checked efficiently and are valid as any SDFG which is inconsistent or deadlocks is not useful in practice.

The SADF model for our LKAS IBC application is illustrated in Fig. 2.3(a). The sensing and processing algorithm receives the camera image frames and detects the regions-of-interest (RoID) in the frames. In this case, the workload is related to the number of regions-of-interest (RoIs). The detected RoIs can be processed in parallel on a multiprocessor platform. The number of allocated processors for our application determines the number of RoI processing (RoIP) actors in our model. In this case, we have two allocated processors and hence two RoIP actors. The total number of RoI detected by RoID determines the workload $w_i$, i.e., $w_i = y_1 + y_2$. The parameters $y_1$ and $y_2$ determine how many RoI need to be allocated to the in-

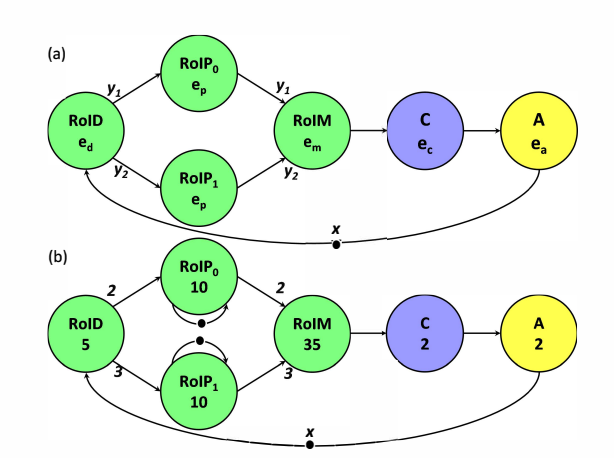



Figure 2.3: LKAS dataflow model, assuming two allocated processors and hence two RoIP actors: (a) application model; (b) (simplified) binding-aware SDFG.

dividual processors and are the rates for the corresponding scenario $s_i$. Note that the workloads here are totally ordered as they are related to the number of RoIs. Rates are annotated with the channels, where rates of 1 are not shown explicitly. The workloads translate to variable token production and consumption rates in the scenario SDFGs. Note that the sensor-to-actuator delay and sampling period vary based on the value of $y_1$ and $y_2$. After processing the RoI, the data is merged and the controller state (the lateral deviation $y_L$ in our LKAS case study) is computed by the RoI merging (RoIM) task. The control algorithm (C) then computes the controller input $u[k]$ (steering angle $\delta_f$ in our LKAS case study) and feeds it to the actuation (A) task.

Each workload $w_i$ in an SADF is associated with an SDFG $\mathcal{G}_i$. An SDFG instance of Fig. 2.3(a) is obtained by assigning values to parameters $e_j$ (the actor execution times) and $y_k$. E.g., assigning $y_1 = 2$, $y_2 = 3$, $e_d = 5$, $e_p = 10$, $e_m = 7\times(y_1+y_2) = 35$, $e_c = 2$, $e_a = 2$ gives the SDFG for a workload of 5 RoIs for mapping to two processors. The actors of $\mathcal{G}_i$ are $\mathcal{A}_i = \{\text{RoID}, \text{RoIP}_0, \text{RoIP}_1, \text{RoIM}, \text{C}, \text{A}\}$. Fig. 2.3(b) illustrates an SDFG as it is derived from the model structure of Fig. 2.3(a). (This is actually a binding-aware SDFG, as explained later.) The set of channels of $\mathcal{G}_i$, $\mathcal{C}_i$, is shown as dependencies in the figure. Compared to the model of Fig. 2.3(a), the SDFG of Fig. 2.3(b) has two additional channels, self-loop channels for the RoIP actors. There are three initial tokens, one on the channel from actor A to RoID and two on the self-loop channels for the RoIP actors. The self-loop channels and their initial tokens capture the fact that each of the RoIP actors is executed sequentially on its allocated processor. The workload scenarios are defined based on $w_i$ and the parameters that change for the corresponding $\mathcal{G}_i$ are $y_1$, $y_2$, and $e_m = 7 \times (y_1 + y_2)$. All other aspects of the $\mathcal{G}_i$ are the same for all scenarios. There is one (labelled) initial token $x$ in the channel from actor A to RoID. For the SADF model in Fig. 2.3, for each scenario $w_i$, the corresponding SDFG $\mathcal{G}_i$ has repetition vector $\rho_i = \begin{bmatrix} 1 & y_1 & y_2 & 1 & 1 & 1 \end{bmatrix}$, where $y_1$ and $y_2$ represent the firing rates of the two actors RoIP shown in Fig. 2.3. A word from the SADF language $\mathcal{F}$ now specifies a sequence of iterations of the corresponding scenario SDFGs.

The state-of-the-art SADF analysis uses (max, +) algebra [9]. The definitions needed for our analysis are summarised in the following paragraphs. For detailed explanations and analysis methods, the reader is referred to [4].

A time-stamp vector $\boldsymbol{\gamma_0}$ captures the availability times of a subset of initial tokens, called the *labelled* initial tokens. The production times $\boldsymbol{\gamma_1}$ of the labelled *final* tokens resulting from the execution of a scenario $s$ are then captured by Eq.( 2.7).

$$\boldsymbol{\gamma_1} = \boldsymbol{G}_s \boldsymbol{\gamma_0}, \tag{2.7}$$

where $\boldsymbol{G}_s$ is the scenario (or state) matrix of $s$. We assume that the labelled initial and final tokens are the same, and that the execution of a scenario corresponds to one iteration of the corresponding SDFG. For the scenario SDFG corresponding to 5 RoI, introduced above in Fig. 2.3(b), the initial token on channel A-RoID labeled

$x$ is a labelled token and $\boldsymbol{\gamma_0} = [0]$. $\boldsymbol{G}_s = \left[e_d + \max(y_1, y_2) \times e_p + e_m + e_c + e_a\right] = [74]$ and $\boldsymbol{\gamma_1} = [74]\,[0] = [74 + 0] = [74]$.

$\boldsymbol{G}_s$ is used to determine the evolution of any scenario sequence. Labelled final tokens of one scenario are the initial tokens of the next scenario execution. E.g., if $s^\omega$ is the infinite repetition of scenario $s$, then the production times of the labelled tokens after the execution of the $k^{th}$ scenario in the sequence are given by:

$$\boldsymbol{\gamma_k} = \boldsymbol{G}_s \boldsymbol{\gamma_{k-1}} = \boldsymbol{G}_s^k \boldsymbol{\gamma_0} \tag{2.8}$$

For all scenarios $s \in \Sigma$, we can construct $\boldsymbol{G}_s \in \mathbb{R}_{-\infty}^{i(s)\times i(s)}$ following the procedure of [119]. Here, $i(s)$ is the total number of labelled initial tokens (in all channels) for scenario $s$ and $\mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$ is the domain of (max, +) algebra.

Further, we need to analyse the production times of *outputs*, i.e., the relevant information produced, during the execution of a scenario sequence. Let the function $m : \Sigma \to \mathbb{N} \cup \{0\}$ map each scenario to the number of outputs produced in that scenario. The output production times of the scenario sequence $s^\omega$ can be computed as,

$$\boldsymbol{p}_k = \boldsymbol{H}_s \boldsymbol{\gamma_k} = \boldsymbol{H}_s \boldsymbol{G}_s^k \boldsymbol{\gamma_0} \tag{2.9}$$

where $\boldsymbol{p}_k$ are the times at which the outputs in the $(k+1)^{th}$ iteration are produced and where $\boldsymbol{H}_s \in \mathbb{R}_{-\infty}^{m(s)\times i(s)}$ is the output matrix of the scenario $s$ that captures the relation between the state vector and the production times of the $m(s)$ outputs. Note that the first output production times are given by $\boldsymbol{p}_0$. The $\boldsymbol{H}_s$ matrices can be computed in a similar way as the state matrices.

For the LKAS scenarios, the output is produced by the actor A, meaning that the output production time is equal to the production time of the token on the channel from A to RoID. This means that $\boldsymbol{H}_s = [74]$ and the production time of the first output $\boldsymbol{p}_0 = [74]\,[0] = [74]$.

We quantify the throughput $\nu$ of a given scenario sequence from the language $\mathscr{F}$ of an SADF model by the average number of outputs produced per time unit during the execution of that sequence. The throughput of an SADF for an infinite scenario sequence $\bar{s}$ is defined as follows.

$$\nu(\bar{s}) = \lim_{n\to\infty} \sup \frac{\sum_{i=1}^{n} m(\bar{s}_i)}{\left\|\boldsymbol{\gamma_n}\right\|} \tag{2.10}$$

where $\bar{s}_i$ refers to the $i^{th}$ symbol in sequence $\bar{s}$ (a scenario), and $\left\|\boldsymbol{\gamma_n}\right\|$ is equal to the maximum entry in the vector $\boldsymbol{\gamma_n}$. For the infinite execution of the 5 RoI scenario SDFG, the throughput is $\frac{1}{74}$. We omit the details of the computation, referring the reader to [4].

For the SADF models in our basic SPADE flow, we assume the following.

- Throughput is inversely monotonic for our SADF model for different workloads. This assumption can be relaxed if we do a brute-force exploration

for system-scenario identification (explained later in Section 2.5.3). This assumption is required as we do not have a DSE step in the basic SPADE flow. The monotonicity is guaranteed by the following:

1. The set of actors $\mathscr{A}$ is the same for all scenarios, i.e. $\mathscr{A}_i = \mathscr{A}_j$, where $\mathscr{A}_i$ and $\mathscr{A}_j$ are the sets of actors of $\mathscr{G}_i$ and $\mathscr{G}_j$ respectively.
2. $w_i \leq w_j \implies \frac{1}{\nu(\mathscr{G}_i)} \leq \frac{1}{\nu(\mathscr{G}_j)}$.

- The sensing task is not pipelined, i.e., the control is sequential. This is guaranteed by having a channel with only one initial token from the actuation task to the start of the sensing task in our SADF model (as in the example).

Both these assumptions are relaxed in Chapter 4.

### 2.3.2 System mapping and mapping configurations

System mapping refers to the binding of the application (modelled as an SADF model) to the given platform (modelled as a platform graph) allocation. Note that for each workload scenario, we can have multiple binding options on the given platform. The throughput of each of these binding options would be different. We then need to find the maximum throughput for a workload scenario, given the platform allocation. The concrete problem is then to find the optimal mapping of a workload scenario to the platform that maximises throughput. Any design flow that does optimal mapping of an application to a platform while maximising throughput can be used. We use the SDF3 design flow [123] as it optimises the resource usage, memory load and communication load for mapping, and embeds state-of-the-art throughput analysis techniques.

Optimal mapping of each workload scenario $s_i$ (modelled as an SDFG $\mathscr{G}_i$) to a platform graph generates a binding-aware SDFG $\mathscr{G}_i^b$ with the task execution schedule encoded in it. The 5-RoI SDFG discussed earlier, and illustrated in Fig. 2.3(b), is in fact a (simplified) binding-aware graph. It captures the binding and scheduling of the actors on two processors. A *mapping configuration* refers to the binding of a workload scenario on the platform and its execution schedule represented as a binding-aware SDFG.

## 2.4 Design problem

We can now make our design problem precise. For a given application and a platform allocation, design

1. mapping configurations,
2. controller configurations, and
3. a runtime reconfiguration mechanism,

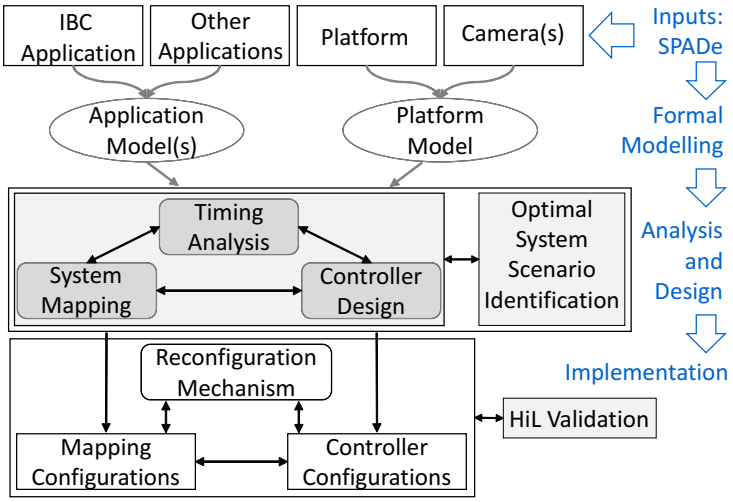



Figure 2.4: Overview of the steps in the basic version of the SPADE flow for parallel implementation (adapted from Fig. 1.8, for readability). In a parallel non-pipelined implementation, the sampling period is not constant and can vary at runtime, as opposed to the proposed SPADE flow for pipelined parallelism outlined in Fig. 1.8.

such that we optimise

- QoC and
- resource utilisation.

## 2.5 Scenario- and platform-aware design (SPADE)

The basic SPADE flow comprises the following steps as shown in Fig. 2.4 (adapted from Fig.1.8):

1. identify, model and characterise the frequently occurring workload scenarios that characterise the dynamic behaviour of the image processing in the control loop;
2. find optimal mappings for these scenarios for the given platform allocation;
3. identify optimal system scenarios combining workload and mapping information and taking into account constraints from the control domain, e.g. stability, and from the embedded domain, e.g. camera frame rate;

4. design a controller with high overall QoC and guaranteed stability for the chosen system scenarios; and

5. a runtime reconfiguration mechanism for implementation.

As already stated, we illustrate the SPADE flow considering the predictability and composibility properties of the CompSOC platform. In the following, we detail the steps in the SPADE flow.



### 2.5.1 SPADE inputs

The inputs to our design flow are details of the IBC application, other applications sharing the platform, given platform allocation for the IBC application and camera characteristics, e.g. FPS. These should be compliant with the application and platform models. Note that the details of the other applications sharing the platform are not relevant for a composable platform such as CompSOC.

### 2.5.2 Formal modelling: application and platform models

A typical IBC application model of an LKAS is shown in Fig. 2.3(a). The details of this model have already been explained in Section 2.3. Task-level WCET profiling is required to compute the WCETs on the CompSOC platform. The platform is modelled as a platform graph as described in Section 1.3.1.

### 2.5.3 Analysis and design

#### System mapping

We first describe the system mapping, i.e., binding and scheduling, of our IBC application model to the platform. Fig. 2.5 illustrates three workload scenarios and their possible platform mapping.[1] Fig. 2.5(a), (c), and (e) model the dataflow graphs for different workloads (note the absence of self-loops for the RoIP actors) and Fig. 2.5(b), (d) and (f) show their corresponding mappings and execution on two or three processor tiles $P_i$. Having more processor tiles means that we can reduce $h$ and $\tau$ by parallel execution of the sensing tasks.

System mapping refers to the mapping of application tasks (modelled as an SADF model) to the platform. An application can have multiple mapping options for a given platform allocation. For example, in Fig. 2.5(c) and (e), the given platform allocation is two and three processor tiles respectively (visible in the number of RoIP actors) for the same workload (5 RoI).

[1] The indices for the RoIP actors are omitted for readability. The functionality of the different RoIP actors is the same.

**Timing analysis: relation between dataflow analysis and control design**

The inverse throughput of the mapped binding-aware SDFG $\mathcal{G}_i^b$ for scenario sequence $s_i^\omega$ gives the sensor-to-actuator delay $\tau_i$; the sampling period $h_i$ can be derived from the sensor-to-actuator delay and expressed in terms of the frame arrival period $f_h$, i.e.,

$$\tau_i = \frac{1}{\nu(s_i^\omega)},\ h_i = \lceil \frac{\tau_i}{f_h} \rceil f_h.$$

The timing parameters for the three mapped workload scenarios in Fig. 2.5 are obtained as follows:

$$\tau_i = e_d + (\max_i^{n_c^{avl}} y_i) \times e_p + e_m + e_c + e_a,\ h_i = \lceil \frac{\tau_i}{f_h} \rceil f_h,$$

where $n_c^{avl}$ represents the total number of available (or allocated) processors. Assume $f_h = \frac{1}{30}$s for a camera with 30 fps and $e_m = 7 \times (\sum_i^{n_c^{avl}} y_i)$. Cost of communicating data between processors is assumed to be part of the actor execution times $e_i$; if meaningful, such cost could be made explicit, but for simplicity, we do not do so. For our example shown in Fig. 2.5:

$$\begin{aligned} \tau_1 &= 5 + 1 \times 10 + 7 \times (1+1) + 2 + 2 = 33\text{ms}, \\ \tau_2 &= 5 + 3 \times 10 + 7 \times (2+3) + 2 + 2 = 74\text{ms}, \\ \tau_3 &= 5 + 2 \times 10 + 7 \times (2+1+2) + 2 + 2 = 64\text{ms}, \end{aligned}$$

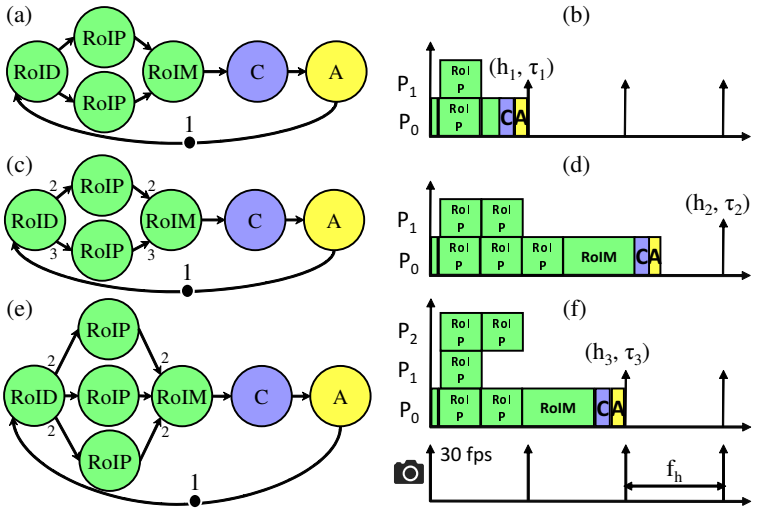



Figure 2.5: Illustration of workload variations and platform mapping.

and $h_1 = f_h,\ h_2 = 3f_h,\ h_3 = 2f_h$.

#### Control design

Once we obtain $\tau_i$ and $h_i$ for mapped workload scenario $s_i$, they are then used for the discrete-time controller implementation as described in Section 2.2 and for designing the controller gains. Further, the timing parameters are a part of the control configuration as defined in Section 2.2.3. Any state-of-the-art control design method can be used for this design.



#### Optimal system-scenario identification

It is possible for multiple workload scenarios to have the same sampling period due to implementation constraints like platform allocation and camera frame rate. For example, for the workload scenario represented in Fig 2.5 (a) with $(h_1, \tau_1)$, the number of ROI, #ROI = 2. However, even for the workload scenario with #ROI = 1 mapped to two processors, we would have the same timing parameters $(h_1, \tau_1)$ since the tasks would have to execute sequentially on one processor. Similarly, for the workload scenario in Fig 2.5 (c), we would have the same timing parameters for #ROI 5 and 6.

A system scenario $s_s$ abstracts multiple workload scenarios $s_i$ such that for $h_s = n \times f_h$ for some $n > 0$, $(h_s - f_h) < h_i \leq h_s$ and $\tau_i \leq \tau_s$. Only system scenarios are then considered for defining the control configuration and for platform implementation.

The system scenarios we consider for this case are $\{s_1 = (f_h, 33),\ s_2 = (2f_h, 57),\ s_3 = s_{wc} = (3f_h, 74)\}\ ms$. The worst-case system scenario $s_{wc}$ is the scenario corresponding to the worst-case image workload (which in our case is 6 ROIs). The other two scenarios correspond to workloads of 1 or 2 resp. 3 or 4 ROIs. The workload with 5 ROIs is part of the worst-case system scenario.

Once we identify the timing parameters for our system scenarios, we design the controller and compute $(K_{s_s},\ F_{s_s})$ for each system scenario $s_s$. Then, the controller stability needs to be guaranteed for the system scenario switching (as explained in Section 2.2.4). If the controller stability is guaranteed, then we can proceed with the implementation. Otherwise, we need to choose a different (sub)set of system scenarios for which the system scenario switching is stable. For the scope of this chapter, this is done by a brute-force approach. In case we cannot identify any stable switching (sub)set of system scenarios, we revert to the design with the worst-case scenario $s_{wc}$ as the single system scenario.

### 2.5.4 Implementation and runtime reconfiguration mechanism

The optimal system scenarios are identified and their corresponding control and mapping configurations are stored as a look-up table (LUT) in platform memory for runtime implementation. During runtime, for every arriving input im-

age frame, we compute the workload (e.g. through an image pre-processing step) and choose the correct system scenario associated with this workload from the LUT. Controller and mapping configurations of the corresponding system scenario are loaded from the LUT. A scheduler then reconfigures the mapping, the time-triggering of the actuation task and the controller gain parameters based on the chosen system scenario. The overhead cost for this reconfiguration has already been considered in our analysis model as a time cost in the start of sensing task (e.g. along with the actor RoID in Fig. 2.3).

## 2.6 Simulation results

In this section, we analyse the performance of our basic SPADE flow through simulations. We first analyse the impact of workload variations and then compare our approach to a state-of-the-art pipelined control implementation.

### 2.6.1 Impact of workload variations

The QoC provided by an IBC system depends on the nature of workload variation encountered by the application resulting in different switching sequences. We simulate the LKAS controller performance for various system scenario switching sequences with 2, 4, and 5 RoIs and sampling periods $h_1 = 0.033$s, $h_2 = 0.066$s, and $h_{wc} = 0.100$s for the corresponding system scenarios $s_1$, $s_2$, and $s_{wc}$, respectively, as shown in Fig. 2.6. We assume that there are 6 RoIs in the worst-case. We observe that our switching designs of SPADE (plot $(s_1 s_2 s_{wc})^\omega$ and $(s_2 s_2 s_{wc})^\omega$) have better QoC (low MSE) than the worst-case sampling period based design (see plot $(s_{wc})^\omega$) in Fig. 2.6. An example switching sequence is illustrated in Fig. 2.7(a).

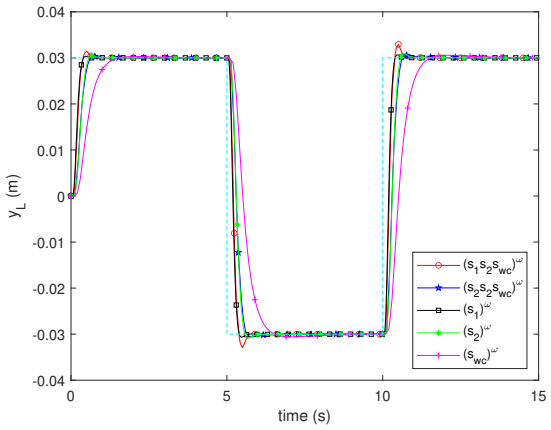



Figure 2.6: Controller performance: comparison of switching subsystems with worst-case $s_{wc}$

We see that the effective resource utilisation for each sampling period is improved (with less idling) with respect to the worst-case based design in Fig. 2.7(b).

### 2.6.2 Comparison with state-of-the-art pipelined control

We compare our SPADE approach with a state-of-the-art pipelined control approach [112]. For fairness in the comparison, we use the same control design technique - LQR with integral action - explained in [112] for SPADE. Further, we consider the same given platform allocation of two processors.

**Pipelined control design:** We discretize the continuous-time system model in Eq. (2.1) with sensor-to-actuator delay $\tau$ and sampling period $h$ to obtain a delayed input system,

$$x((k+1)h) = A_d x(kh) + B_d u(kh-h), \tag{2.11}$$

where $A_d$, $B_d$ are the discretized state and input matrices respectively. Here, $A_d = e^{A_c h}$ and $B_d = \int_0^h e^{A_c s} B_c ds$. The control input $u(t)$ applied at $t = kh$ uses $h$ time units old sensing information in any sampling interval $kh$ to $(k+1)h$ due to the sensor-to-actuator delay $\tau$. This is reflected in Eq. (2.11) as the delayed input $u(kh-h)$.

For brevity, the pipelined control delay and period is represented as $\tau$ and $h$ in this subsection. $\tau = \lceil \frac{e_{\text{total}}}{f_h} \rceil f_h$ and $h = \frac{\tau}{\gamma}$, where $\gamma$ is the number of processing cores. Note that in [112], there is a strict criterion that the sampling period should be an integral multiple of $f_h$ and strictly periodic. As such the $\tau$ should be relaxed based on $\gamma$. E.g., in our LKAS case, if $e_{\text{total}} = 0.084$s, we get $\tau = 0.100$s and $h = 0.050$s for $\gamma = 2$. However, $h = 0.050$s is not an integral multiple of $f_h$ and as such we have to relax $\tau$ so that $\tau = 0.100 + f_h = 0.133$s and $h = 0.067$s which is an integral multiple of $f_h$.

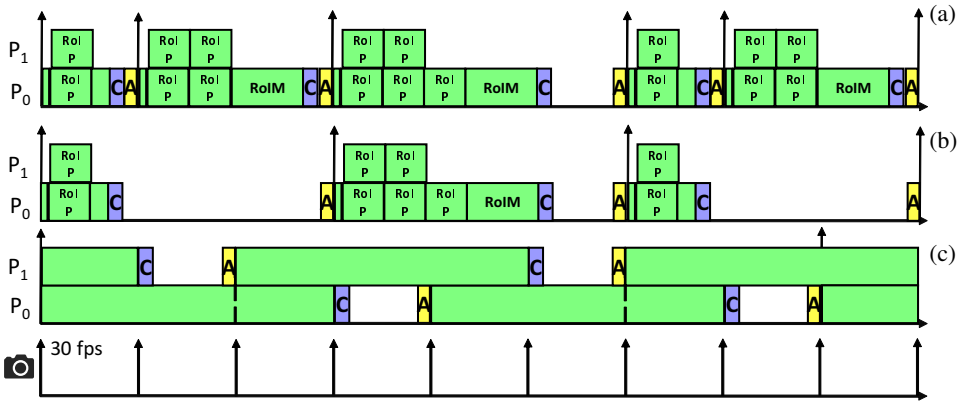



Figure 2.7: Gantt charts for (a) switching sequence $(s_1 s_2 s_{wc})^\omega$, (b) corresponding worst-case design $(s_{wc})^\omega$, and (c) pipelined control design used for comparison.

For designing the delayed control input $u(kh-h)$, one design option is to transform the system in Eq. (2.11) into standard non-delayed form and apply any standard control design technique. Towards this, we define a new system state vector $\hat{z}(kh) = \begin{bmatrix} x^T(kh) & u(kh-h) \end{bmatrix}^T$ to obtain a higher-order augmented system in the non-delayed form as follows:

$$\begin{aligned} \hat{z}(kh+h) &= \Phi_d \hat{z}(kh) + \Gamma_d u(kh) \\ y(kh) &= C_d \hat{z}(kh), \end{aligned} \tag{2.12}$$

where $\Phi_d$, $\Gamma_d$, $C_d$ are the augmented discretized matrices such that,

$$\Phi_d = \begin{bmatrix} A_d & B_d & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \Gamma_d = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 1 \end{bmatrix}.$$

$\mathbf{0}$ and $\boldsymbol{I}$ represent the zero and identity matrices of appropriate dimensions. A check for controllability [37] is done for this augmented system. If the system is not controllable, controllability decomposition is done to obtain a controllable subsystem.

The system in Eq. (2.12) is in standard discrete-time form for which a standard discrete-time control design technique such as LQR [37] can be used. We use an LQR-based optimal state feedback controller, with integral action for reference tracking, referred to in literature as linear-quadratic-integral (LQI) control [142, 144]. The state feedback controller is of the form,

$$u(kh) = K \begin{bmatrix} \hat{z}(kh) \\ x_i(kh) \end{bmatrix}, \text{ where } x_i(kh+h) = x_i(kh) + y(kh) - r(kh). \tag{2.13}$$

$K$ is the LQR state feedback gain designed for the state space considering the integral action as given below,

$$\begin{bmatrix} \hat{z}(kh+h) \\ x_i(kh+h) \end{bmatrix} = \begin{bmatrix} \Phi_d & \mathbf{0} \\ C_d & \mathbf{1} \end{bmatrix} \begin{bmatrix} \hat{z}(kh) \\ x_i(kh) \end{bmatrix} + \begin{bmatrix} \Gamma_d \\ \mathbf{0} \end{bmatrix} u(kh). \tag{2.14}$$

This control design replaces the earlier presented LQR control design for each scenario in SPADE. We do so to have a fair comparison between different implementation strategies since the control theory for pipelined control systems considers that $\tau$ and $h$ are integral multiples of $f_h$. However, the controller design approach of SPADE can have any value for $\tau$ and hence is more flexible than pipelined control. We adapt the pipelined control design applicable for $\tau \geq h$ to the SPADE approach applicable for any $\tau < h$ by modifying Eq. (2.14) with parameters from Eq. (2.4):

$$\begin{bmatrix} \hat{z}(kh+h) \\ x_i(kh+h) \end{bmatrix} = \begin{bmatrix} A_{aug,s_s} & \mathbf{0} \\ C_{aug} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \hat{z}(kh) \\ x_i(kh) \end{bmatrix} + \begin{bmatrix} B_{aug,s_s} \\ \mathbf{0} \end{bmatrix} u(kh).$$

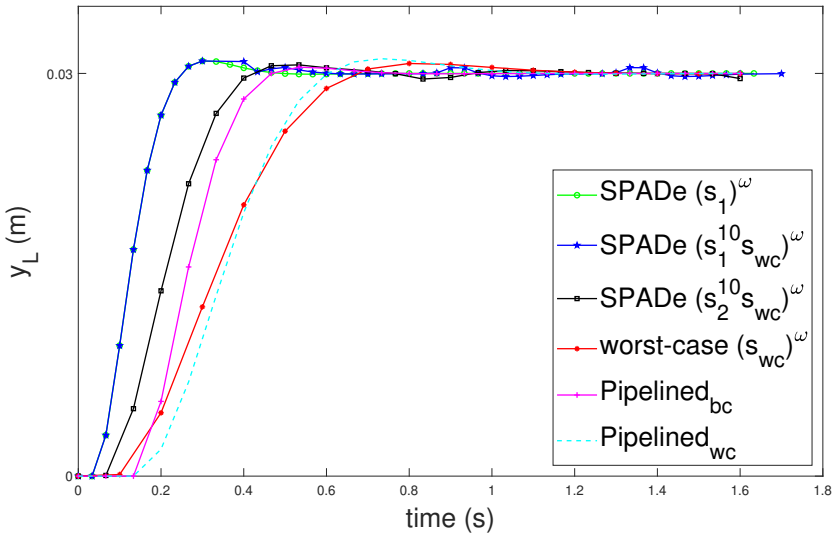



Figure 2.8: Comparison between pipelined and SPADE approach

Then, we design the gain $K_{s_s}$ for each SPADE scenario $s_s$ using Eq. (2.13).

**Comparison:** For pipelined control, the total sensor-to-actuator delay $e_{\text{total}} = 5+6\times10+6\times7+2+2 = 111$ms (since each pipe executes sequentially), the effective sensor-to-actuator delay = $\tau = \lceil \frac{e_{\text{total}}}{f_h} \rceil f_h = 133.33$ms, and the sampling period $h = \frac{\tau}{2} = 66.67$ms for the given two processing cores. The Gantt chart of the pipelined execution is shown in Fig. 2.7(c). As mentioned, for SPADE, we use the same pipelined control design approach. For scenario $s_{wc}$, $e_{\text{total}} = 5+3\times10+6\times7+2+2 = 81$ms so that $\tau_{s_{wc}} = 0.100 = h_{wc}$. Similarly, for scenario $s_1$, $\tau_1 = 0.033 = h_1$ and for scenario $s_2$, $\tau_2 = 0.067 = h_2$.

The results of the comparison between the pipelined controller with respect to the SPADE approach are shown in Fig. 2.8. Note that SPADE allows for parallelisation that reduces both sampling period and sensor-to-actuator delay. However, pipelining only reduces the sampling period.

The key observations are:

- The performance of the LQI controllers highly depends on the quality of controller tuning [112]. We observe that the QOC of the pipelined controller is always in the range of QOC between the worst-case design and the SPADE approach. Fig. 2.8 shows two different tunings of the pipelined controller: plot pipelined$_{\text{bc}}$ is tuned with the same control parameters as scenario $s_1$ and pipelined$_{\text{wc}}$ is tuned with the same control parameters as scenario $s_{wc}$.

  If we execute in a frequently occurring scenario, e.g., $s_1$ (see plot $(s_1^{10}s_{wc})^{\omega}$ in Fig. 2.8), then we see that the control performance is better than the pipelined control. In this particular case, arbitrary switching between $s_1$, $s_2$,

Table 2.1: SPADE vs pipelined: applicability criteria and comparison

| Criteria | SPADE | Pipelining [112] |
|---|---|---|
| Algorithm | should be white/gray box | white/gray/black box |
| Degree of parallelisation | should be high for better QoC | independent (no parallelism) |
| Inter-frame dependencies | independent (no pipelining) | should not exist |
| Workload variations | considered in design | not considered |
| Platform | independent (applicable for all) | suitable mainly for homogeneous |
| Restrictions on $h$ | any multiple of $f_h$ | multiple of $f_h$; strictly periodic |
| Restrictions on $\tau$ | none | multiple of $h$ and $(\gamma \times f_h)$ |



and $s_{wc}$ is unstable. To meaningfully apply the SPADE approach, we should have a frequently occurring scenario during run time, and switching from this frequently occurring scenario to the worst-case should be stable, e.g., based on a dwell time criterion [125] (as shown for plots $(s_1^{10} s_{wc})^\omega$ and $(s_2^{10} s_{wc})^\omega$ in Fig. 2.8).

- SPADE performs better with a shorter $\tau$ when $\tau < h$ and other control tuning parameters are kept the same. When $\tau_1 < \tau_2 < h$, the case with $\tau_1$ will have better performance than $\tau_2$ for the same $h$. The actual performance improvement further depends on the system dynamics.

SPADE gives prominent advantages when the algorithm structure is known, i.e., the application is a white/gray box and when there is scope for parallelisation. Pipelining works also when the application is a black box and is not dependent on the parallelisation of the algorithm. However, pipelining cannot be used when there are inter-frame dependencies for the algorithm, whereas SPADE is not affected by inter-frame dependencies (as further elaborated in the next two chapters). Further, SPADE gives better results when the application is executing in its frequently occurring scenario. Pipelining is better suited if the application is frequently executing closer to its worst case. A brief comparison between SPADE and pipelined approaches is illustrated in Table 2.1. In the following two chapters, we integrate pipelining in SPADE, effectively combining the advantages of the two approaches.

## 2.7 SPADE for an industrial platform

### 2.7.1 Case study: multi-camera LKAS sharing the platform with other applications

We consider a concrete case study of a multi-camera LKAS. The goal of the LKAS is to steer the vehicle autonomously to follow the center line of a lane. Multiple cameras are used since the field-of-view of a single camera is not sufficient to detect the lanes when the vehicle has to make sharp turns, e.g., at a T-junction. Fig. 2.9(c) and (d) show the two different scenarios in the LKAS system. The first scenario $s_1$

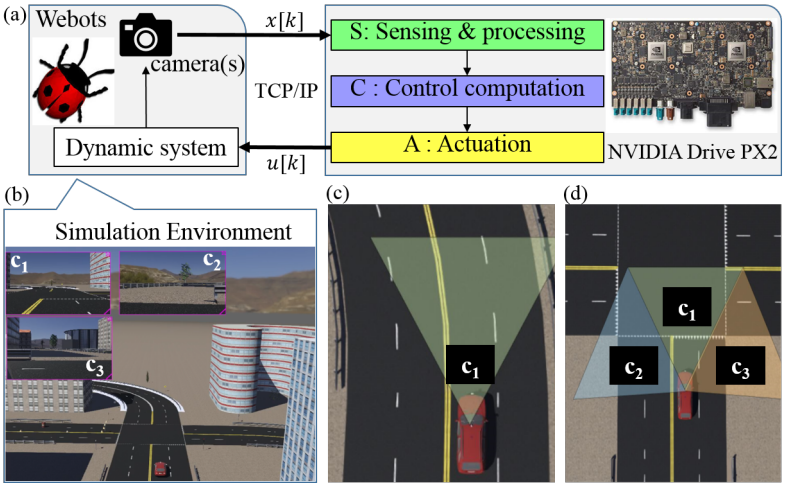



Figure 2.9: (a) IBC system block diagram and the HIL simulator. (b) a snapshot of the HIL simulation environment in webots. (c) LKAS using single camera. (d) multi-camera LKAS; $c_1$, $c_2$, $c_3$ are the cameras.

(see Fig. 2.9(c) occurs when the vehicle is navigating on a road with no sharp turns. In scenario $s_1$, only one camera $c_1$ needs to be active. The second scenario $s_2$ (see Fig. 2.9(d) happens when the vehicle needs to take a sharp turn. In this case, all three cameras $c_1$, $c_2$ and $c_3$ need to be active. During runtime the scenarios are detected based on the following: i) when there is a lane detected by camera $c_1$ and there is no request to make a turn, the LKAS executes in scenario $s_1$; ii) when there is no lane detected by camera $c_1$ or there is a request to make a turn, the LKAS executes in scenario $s_2$. Our multi-camera LKAS is sharing the NVIDIA Drive PX2 platform with two other data-intensive applications - object detection and tracking (ODT) and automatic emergency braking (AEB).

### 2.7.2 SPADE input: IBC application

**Image sensing and processing (*S*)**

The main stages in the compute-intensive image sensing and processing of an automotive IBC system are the image-signal processing (ISP) pipeline, environment perception and application-specific rendering (if required) (shown in Fig. 2.10(a)). The ISP pipeline is generic for all IBC applications. Environment perception involves application-specific preprocessing, feature extraction and inference. Rendering refers to the display of relevant information on the dashboard or screen of the vehicle and is application-specific. Below, we explain these stages in detail for our LKAS system case study.

**Image-signal processing (ISP) pipeline** The NVIDIA Drive PX2 comes with a Tegra configurable ISP hardware and supports different image types - CUDA, OpenGL, NvMedia - and different pixel formats - RAW, grayscale, RGB, Red Clear Clear Blue (RCCB), RGB alpha (RGBA), YUV. NvMedia is an NVIDIA proprietary framework which uses dedicated hardware blocks on the Tegra system-on-chips (SoCs) for faster image processing. Algorithmic analysis of a closed-source proprietary ISP pipeline is not possible. The stages common to generic ISP pipelines are explained in [21]. For our LKAS, the gigabit multimedia serial link (GMSL) camera [97] captures the image frame at a fixed frame rate, 30 fps. Each frame then goes through the closed-source ISP pipeline to obtain an image in ≪NvMedia, YUV≫ format.

**Perception** The perception stage performs a set of application-specific preprocessing, feature extraction and control state computation steps on the image obtained from the ISP.

The preprocessing step in LKAS (shown in Fig. 2.10(a)) involves converting the image in ≪NvMedia, YUV≫ format to the ≪CUDA, RGBA≫ and ≪OpenGL, RGBA≫ image type and pixel formats. Closed-source functions 'image streamer' and 'format conversion' from NVIDIA perform the image type conversions and pixel format conversions, respectively. The ≪CUDA, RGBA≫ format is used for applications that use graphical processing units (GPUs) and ≪OpenGL, RGBA≫ for rendering.

The features to extract are application-specific. The LKAS extracts the lanes from the image using the NVIDIA proprietary (pre-trained) high-precision deep neural network (DNN) *Lanenet* [137] that enables pixel-level lane detection. Lanenet executes on the GPU and its input is a ≪CUDA, RGBA≫ image. The output of Lanenet is the position values of all the lane containing pixels, i.e., a set of polyline values in the pixel domain.

Finally, the lateral deviation of the vehicle from the center of the lane is derived. A homography transformation matrix [56] is computed at design time. This matrix is stored in the platform memory and is used at runtime to compute the position values of the detected polylines from Lanenet. The left and right lane polylines are then fit to a second degree polynomial. For a given look-ahead distance, the center of the lane is derived using these polynomials while the center of the image gives the vehicle's current position. Using these, the lateral deviation is calculated at the look-ahead distance. The homography transformation at runtime needs to be done only for the identified lane pixels.

**Rendering** For LKAS, the rendered image consists of the pre-processed image captured by the camera in ≪OpenGL, RGBA≫ format superimposed with the polylines detected by Lanenet. The rendering step is not important for the correct functioning of LKAS. Rendering is used for debugging and often provided as an add-on for automotive customers for visual pleasure.

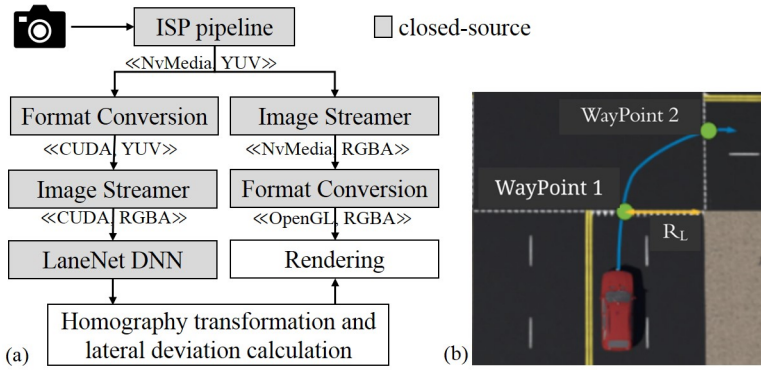



Figure 2.10: (a) The block diagram of image sensing and processing task $S$. (b) Path planning for scenario $s_2$.

**Control computation ($C$) and actuation ($A$) tasks**

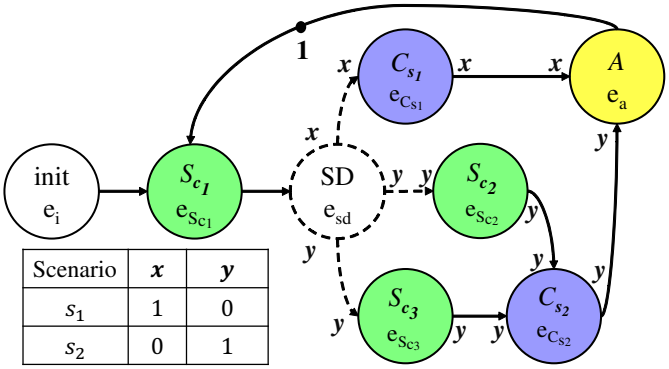
The default **scenario $s_1$** persists when there is always a lane detected in the image captured by the camera $c_1$ and when there is no request to take a turn, e.g., at a junction. For this scenario, the LKAS controller explained in Section 2.2 is used. **scenario $s_2$** occurs when there is no lane detected by camera $c_1$ or when there is a request to take a sharp turn. Here the control computation is a standard path planning algorithm. The direction of the turn is user input or determined arbitrarily if lanes are detected on both $c_2$ and $c_3$. If no lanes are detected in any of the cameras, AEB is activated. Actuation task $A$ actuates the vehicle steering to the desired value communicated to it by the control computation task.

### 2.7.3 Formal modelling

The application is modelled as a dataflow graph (see Fig. 2.11 and Section 2.3). The applications sharing the platform act as load and the platform is modelled as a platform graph (see Fig. 1.6) using the available information [97] (see Section 2.2).

The LKAS has two application scenarios. The *init* actor models platform initialisation. The $S_{c_1}$, $S_{c_2}$ and $S_{c_3}$ actors model the image sensing and processing tasks for cameras $c_1$, $c_2$ and $c_3$. Each sensing task has the internal structure shown in Fig. 2.10(a). $C_{s_1}$ and $C_{s_2}$ model the control computations for scenarios $s_1$ and $s_2$. The actuation task is modelled by actor $A$. The scenario detector actor $SD$ determines which scenario the application runs in. $e_i$, $e_{S_{c_1}}$, $e_{sd}$, $e_{C_{s_1}}$, $e_{S_{c_2}}$, $e_{S_{c_3}}$, $e_{C_{s_2}}$ and $e_a$ are the execution times of the corresponding actors.

Note that the workload for this case is defined by the combination of application scenarios, non-predictable timing behaviour of the closed-source industrial platform and the platform load. Each platform load condition is abstracted as a variant (see Table 2.2) for a systematic analysis. The parameters that change based




| Scenario | $x$ | $y$ |
|---|---|---|
| $s_1$ | 1 | 0 |
| $s_2$ | 0 | 1 |

Figure 2.11: Multi-camera LKAS SADF model.

on the workload are $x$, $y$ shown in Fig. 2.11 (due to the application scenarios) and the execution times of all actors (due to the platform load).

### 2.7.4 Analysis and design

#### System mapping and mapping configurations

The tasks/actors need to be mapped to the Tegra SoC resources - central processing unit (CPU), integrated Pascal GPU (iGPU), discrete GPU (dGPU) and memory, where CPU refers to the A57 and denver2 cores. Note that there are two Tegra SoCs in the NVIDIA Drive PX2 platform. The options available for the developer when mapping to the PX2 platform are limited. Priorities for tasks can only be assigned on CPU resources. The tasks mapped to the GPUs are executed by the proprietary NVIDIA scheduler and the execution times for such tasks vary the most due to the (unpredictable) scheduler [140].

The mapping configuration does not include the schedule of tasks mapped to GPUs as it is not controllable. Note that init, the homography computation step of $S$, $SD$, $C$ and $A$ are always mapped to CPUs and the other steps of $S$ are always mapped to GPUs. The GPUs can only be accessed through the CPUs in the same SoC by a blocking call.

#### Profiling and timing analysis

Platform-aware profiling is a crucial step in this instance of the SPADE flow. Since there are closed-source functions in the application and a non-real-time Ubuntu OS, the WCETs of tasks in the application are difficult to predict. The WCET of tasks depends on three factors: scenario of the IBC application, choice of mapping, and the load on the platform due to the shared applications. For our case study, we consider two other applications - ODT and AEB - sharing the platform.

Table 2.2: Characteristics of variants based on mapping choice and load conditions

| variants | v.1 | v.2 | v.3 | v.4 | v.5 | v.6 | v.7 | v.8 | v.9 | v.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| mapping | DGPU | IGPU | DGPU | IGPU | DGPU | IGPU | DGPU | IGPU | DGPU | IGPU |
| load | - | - | AEB | AEB | ODT | ODT | AEB, ODT | AEB, ODT | ODTs | ODTs |

ODTs: Object detection and tracking with task sharing between GPUs



Both applications take camera images as input.

We define variants $v.i$ to characterise and abstract multiple workload scenarios for a structured profiling and timing analysis. The definition of variants is not a necessity for the basic version of the SPADE flow. However, it helps to classify the expected runtime scenarios and perform profiling and timing analysis in a structured way for the designer. The variants we consider based on our mapping choice and platform load are defined in Table 2.2. The mapping is characterised based on mapping the LKAS application to the IGPU or the DGPU, as preliminary experiments show that compute-intensive imaging tasks perform better on GPUs. Tasks mapped to CPUs take less than 5% of the overall WCET and are not explicitly considered. The platform loads AEB and ODT denote the mapping of these applications to the same GPU as the LKAS. The platform load ODTs denotes the mapping of ODT and LKAS to multiple GPUs (of the same type) so that there is task sharing between GPUs. This can be done in the NVIDIA platform by assigning just the type of GPU for the applications; then the proprietary scheduler allocates tasks between multiple GPUs. This can be observed by analysing the GPU utilisation (explained in Section 2.7.7).

Note that due to the closed-source GPU scheduler of NVIDIA, the workload due to the application scenarios and the platform load conditions at runtime cannot be distinguished. Thus, the abstraction as variants is a means to enable the optimal system-scenario identification and runtime reconfiguration (explained in Section 2.7.5).

For profiling, a database of around 200 images (captured by the GMSL camera) is identified with varying image workload. Considering image workload variations is important since they affect the WCET analysis. The image for the minimal workload has no lane markings and no other vehicles on the road; for the maximal workload, it contains three lane markings and other vehicles. For each variant, each image from the database is run on the PX2 for 10000 iterations.

The worst-case sensor-to-actuator delay $\tau_{wc}$ and sampling period $h_{wc}$ are computed as explained in Section 2.5.3. The execution times of each task are profiled over all the variants and the maximum value is taken as the estimated WCET of the corresponding task. This WCET estimate is used in the application SADF model. $\tau_{wc}$ and $h_{wc}$ are then computed. Note that though this worst-case rarely happens, it is needed to guarantee stability of the IBC system. Similarly, for each variant $v.i$, the third quartile values of the measured execution times of each task (profiled for the corresponding variant) is used to compute $\tau_i$ and $h_i$. We thus

avoid the measured WCETs for the majority of the analyzed workload scenarios to avoid overly pessimistic model predictions.

**Controller design**



The controller for scenario $s_1$ is designed as explained in Section 2.2. The standard linear quadratic regulator control is used to design the state feedback gain $K_i$ and the feed forward gain $F_i$ for each variant $v_i$. The control configuration $\chi^c_{s_i}$ is then defined as a tuple $\chi^c_{s_i} = (\tau_i, h_i, K_i, F_i)$. For each version, only $\chi^c_{s_i}$ needs to be stored in the memory during implementation. The stability of this switched system is analysed by deriving LMIs that check for the existence of a CQLF (see Section 2.2.4).

For scenario $s_2$, the path planning algorithm identifies two waypoints once the direction to turn is determined (illustrated in Fig. 2.10(b) for a 90 degree turn). Waypoint 1 is the centre of the lane from where the vehicle has to start turning and Waypoint 2 is the centre of the lane after the turn, from where we expect scenario $s_1$ (see Fig. 2.10(b)). This can be predicted based on the turning radius $R_L$. The steering angle $\delta_f = atan\left(\frac{L_{wb}}{R_L}\right)$, where $L_{wb}$ is the wheelbase of the vehicle. This steering angle is constantly applied from Waypoint 1 until the vehicle reaches Waypoint 2 and then task $S$ repeats. Only $L_{wb}$ needs to be stored in memory for scenario $s_2$.

### 2.7.5 System-scenario identification, implementation and runtime reconfiguration mechanism

The variants defined in this section classify the expected runtime scenarios (as explained in Table 2.2). The variants are useful in the profiling and timing analysis step in the basic SPADE flow. A system scenario abstracts multiple variants with the same sampling period and optimal system scenarios are identified as explained in Section 2.5.3. During runtime, we keep track of the start and finishing time of $S$, i.e., the sensing delay, to check for which system scenario we need to execute from the LUT. The control and mapping configurations of the variants and their relation to system scenarios are stored as a LUT in platform memory for runtime implementation. After identifying the system scenario, we load the corresponding control configuration $\chi^c_{s_i}$ and execute $C$. The mapping configuration is then loaded for the subsequent arriving frame. Note that even though control configurations are loaded every frame, mapping configurations cannot be loaded until after the system scenario identification is completed and as such there is a delay in loading mapping configuration by one frame. The classification as variants is thus essential in the identification of the system scenario at runtime as the scenario identification at runtime is dependent on the current mapping as well. An LQR controller designed for the worst-case $(\tau_{wc}, h_{wc})$ and its corresponding control configuration $\chi^c_{s_{wc}}$ is also stored in the memory as the worst-case system scenario. At runtime, system scenarios are switching (as explained in Section 2.7.4)

based on the load and/or mapping choices.

### 2.7.6 Design-space exploration at design time: QoC vs utilisation trade-offs

The QoC is defined as the inverse MSE (defined in Section 2.2.5) so that a lower MSE means a better QoC. The utilisation at design time is defined based on the estimated time spent by the application in GPU kernel calls. We use this definition for the utilisation metric as this could be computed in an actual implementation as well. A DSE to obtain different system configurations is performed for each of the variants defined in Table 2.2 by choosing different mapping options for $S$, $C$ and $A$ tasks. Note that the task mappings are allowed to span over two Tegra SoCs and the subtasks of $S$ (shown in Fig. 2.10(a)) can also be mapped to separate GPUs. Pareto-optimal system configurations are then identified for each variant through Pareto optimisation. For each variant, DSE, profiling and timing analysis provide different configurations with variable task execution times. This results in variable delay for each variant and hence multiple system configurations with variable sampling period due to the characteristics of the industrial platform. As a basis for system-scenario identification, we select the Pareto-optimal system configurations for each variant with delay and sampling period derived from the configuration with frequently occurring task execution times.

The $\tau_i$ and $h_i$ from these Pareto-optimal configurations for each variant are shown in Table 2.3. The $v.i$ in Fig. 2.12 correspond to the predicted QoC vs. utilisation design points using our design flow. In Fig. 2.12, the MSE for $v.8$ with the largest $\tau_i$ among different variants is the poorest. The MSE for different variants tends to aggregate based on the $h_i$. System scenarios can then be identified based on the requirements, e.g. if QoC is the only criterion, we can select variants $v.1$, $v.7$, and the worst case as system scenarios.

The pessimistic $\tau_{wc}$ and $h_{wc}$ are estimated as explained in Section 2.5.3 to be 0.150 s and 5/30 s respectively. Note that $h_7$ is only 2/30 s and the identified worst-case variant $v.8$ has $h_8 = 0.100$ s. This means that a control design for $h_{wc}$ would see a much worse MSE than any of the variants.

### 2.7.7 Hardware-in-the-loop validation using NVIDIA Drive PX2

A design-time analysis alone is insufficient as the runtime behaviour of an industrial platform cannot be predicted. We implement the 10 different variants mentioned in Table 2.2 using a HiL simulator for LKAS (shown in Fig. 2.9) and compare its performance with the design-time analysis. Our HiL simulator uses webots [84] as the physics simulation engine and interacts with NVIDIA Drive PX2 using the transmission control protocol/internet protocol (TCP/IP). This setup uses webots for multi-camera LKAS with support for turning at a junction (or at user input). This setup is part of the performance evaluation for IMAge-based Control Systems (IMACS) framework (a contribution of this thesis).

Table 2.3: Bounded $\tau_i$ and $h_i$ of selected Pareto-optimal system configurations (corresponding to the frequently occurring task execution times) per variant, providing the basis for the definition of system scenarios.

| variants | v.1 | v.2 | v.3 | v.4 | v.5 | v.6 | v.7 | v.8 | v.9 | v.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\tau_i$ in s | 0.028 | 0.042 | 0.034 | 0.066 | 0.040 | 0.069 | 0.051 | 0.100 | 0.038 | 0.045 |
| $h_i$ in s | 1/30 | 2/30 | 2/30 | 2/30 | 2/30 | 3/30 | 2/30 | 3/30 | 2/30 | 2/30 |



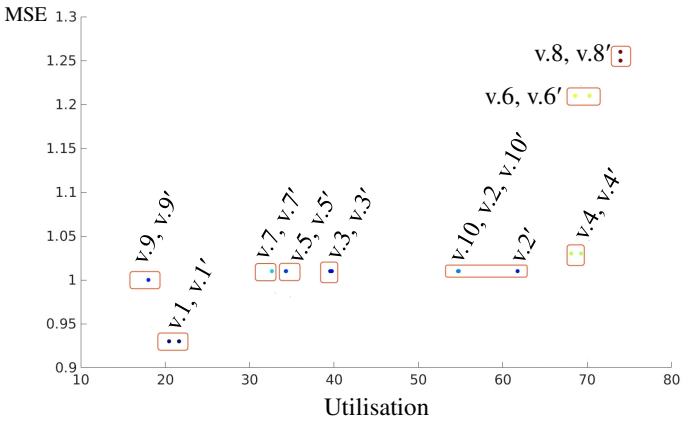



Figure 2.12: Validating the design time Pareto-optimal system configurations for each variant $v.i$ with the corresponding HiL implementation $v.i'$.

The performance metrics we consider are MSE (explained in Section 2.2.5) and GPU utilisation. GPU *utilisation* is measured by the proprietary NVIDIA Nsight software [99]. GPU utilisation gives the measure of the time spent by the application in GPU kernel calls. For compute-intensive image-based applications sharing the platform, minimising the utilisation is better.

The $v.i$ in Fig. 2.12 correspond, as already explained, to the predicted design points using our design flow and the $v.i'$ correspond to the design points obtained from actual implementation using the HiL simulator. Even though the numbers vary between our design flow prediction and actual implementation, the trends we observe for the different variants are the same. Recall that we used measured third quartile execution times instead of WCET in our models. At run time, when we encounter the WCET or any violation of the $(\tau_i, h_i)$ for $v.i$, we execute the worst-case controller designed for $(\tau_{wc}, h_{wc})$. At runtime, a switched controller considering the different variants has a much better MSE than the worst-case as there is no aggressive switching, i.e. once we are running in a particular variant, the runtime situation persists for some time. Notice that the QoC improves at runtime since the controller executes in the frequently occurring system scenario.

## 2.8 Conclusions

We introduced the SPADE approach with parallelization as a structured IBC (co-) design flow. The presented SPADE approach considers sensing-application parallelism, workload variations, platform settings, and control parameters for an efficient design and implementation of an IBC system. Our SPADE approach optimises control quality and maximises the effective resource utilisation for a given platform allocation. We demonstrate the applicability of SPADE for both a predictable multiprocessor platform and for an industrial platform. Though application timing is difficult to predict in industrial platforms, we show that we can leverage existing predictable dataflow model-based design methods by carefully co-designing the sensing implementation and the (switched) controller design using system scenarios.


3

# SPADE by pipelining

3

Image-based control (IBC) systems have a long sensing delay. The advent of multiprocessor platforms helps to cope with this delay by pipelining of the sensing task. However, existing pipelined IBC system designs are based on linear time-invariant models and do not consider constraint satisfaction, system nonlinearities, workload variations and/or given inter-frame dependencies, which are crucial for practical implementation. A pipelined IBC system implementation using a model-predictive control (MPC) approach that can address these limitations making a step forward towards real-life adoption is thus promising. We present an adaptive scenario- and platform-aware design (SPADE) MPC formulation based on linear parameter-varying input/output models for a pipelined implementation of IBC systems. The proposed method maximizes quality-of-control by taking into account workload variations in the image processing for individual pipes in the sensing pipeline in order to exploit the latest measurements, besides explicitly considering given inter-frame dependencies, system nonlinearities and constraints on system variables. The practical benefits are highlighted through simulations using vision-based vehicle lateral control as a case study (already introduced in Chapter 1). In this chapter, the SPADE approach focuses on pipelining without parallelising the sensing task. Chapter 4 focuses on pipelined parallelism.

## 3.1 Background and contributions

IBC systems refer to a class of data-intensive feedback control systems whose feedback is provided by camera sensor(s) (see Fig. 3.1). The combination of camera sensor(s) and image processing algorithms is capable of detecting a rich set of features in an image that can be used to compute the states of the system such as relative position or distance, depth perception, and tracking of the object-of-interest [101]. The challenge, however, is that there is an inherent long sensing delay due to compute-intensive image processing algorithms [110].

A typical implementation of an IBC system uses an optimal linear quadratic regulator (LQR) [37] considering the worst-case image workload and thus has

The content of this chapter is an adaptation of the following paper:
Sajid Mohamed, Nilay Saraf, Daniele Bernardini, Dip Goswami, Twan Basten, and Alberto Bemporad. Adaptive predictive control for pipelined multiprocessor image-based control systems considering workload variations. In *59th IEEE Conference on Decision and Control (CDC)*, 2020.

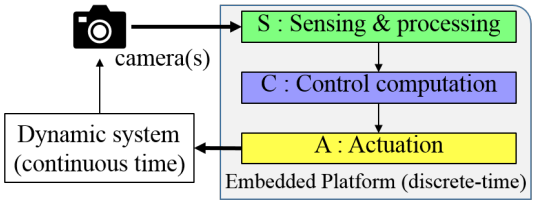



Figure 3.1: An IBC system: block diagram (repeating Fig. 1.4 (a), for readability)



worst-case sensing delay [110] (illustrated in Fig. 3.2). However, this results in poor effective resource utilisation in a multiprocessor platform, and suboptimal quality-of-control (QoC) [46]. Multiprocessor platforms with high processing power that allow parallel and pipelined executions can be used to cope with this long worst-case sensing delay. The sensing algorithms may be parallelised (whenever possible, but limited by the degree of application parallelism) if the algorithmic structure is known (white/grey box) and thus reduce the worst-case sensing delay [110]. Pipelined control implementation executes the sensing algorithm in a pipelined fashion with the worst-case sensing delay and thus reduces the effective actuation and sampling rate [112], [69]. The advantage of a pipelined implementation is that it is applicable even if the application algorithm is a black box. Note that for nominal control implementation, the sensor-to-actuator delay $\tau$ is at most equal to the sampling period $h$, whereas for a pipelined control implementation $\tau$ is greater than $h$.

However, pipelining is limited by inter-frame dependencies, i.e., the data or algorithmic dependencies between consecutive frame processing, e.g., due to video coding [77] or visual tracking [120]. Inter-frame dependence time (denoted by $f_d$) can be quantified for the current image frame as the maximum time required to complete the processing of (parts of) the algorithm the subsequent image frame processing depends on. Alternatively, $f_d$ is the maximum time required to wait between processing consecutive image frames.

The sensing delay due to the compute-intensive processing (sensing) of the image stream is dependent on image workload variations, which occur due to image content and result in a wide range between best-case and worst-case image-processing times. It is known from [46] that explicitly taking into account workload variations in controller design improves the QoC. Workload variations are typically considered as a variable delay or stochastic delay in standard sampled-data linear control design techniques [100].

In current literature, workload variations are typically considered only for sequential IBC implementation [46] and not for pipelined implementation [112], [69] (see Fig. 3.3). In [82], pipelining is considered along with variable delay but the inter-frame dependencies are neglected. Further, these approaches do not consider system nonlinearities, i.e., the variations in system dynamics, and constraints imposed on the system variables, which can be crucial when considering a practical implementation. E.g., the maximum steering angle of the Udacity

self-driving car is set to +/- 25 degree [132] and the vehicle velocity is kept constant for the simulations in [68].

The main motivation of this work is to address limitations of the state-of-the-art pipelined multiprocessor IBC approaches that do not take into account inter-frame dependencies, system constraints and nonlinearities for the application of interest. These limitations make it difficult for these approaches to be realised in real systems.

**Contribution:** We extend the basic SPADE flow (introduced in Chapter 2) for IBC system design for pipelined implementation without parallelising the sensing task. Our approach explicitly takes into account the inter-frame dependencies. Further, we present an adaptive predictive control formulation based on linear parameter-varying (LPV) input/output (I/O) models for a pipelined multiprocessor implementation of IBC systems while considering workload variations, system nonlinearities and constraints on system variables, and thus makes a step forward towards real-life adoption. We also compare the proposed formulation with the state-of-the-art multiprocessor IBC system implementations.

Recent advances in numerical optimization for MPC have enabled safety-critical applications on embedded platforms, such as engine control and power-train coordination in the automotive domain [13,14]. Moreover, the latest methods such as those recently reported in [115] suggest that the model and MPC tuning parameters can be adapted at runtime without reconstructing the optimization problem. This allows implementing adaptive MPC with the same computational complexity as the non-adaptive case. An MPC formulation is thus advantageous for use in image-based control systems where, due to constraints, nonlinearities and workload variations, an adaptive control method that maximizes control performance is desirable.

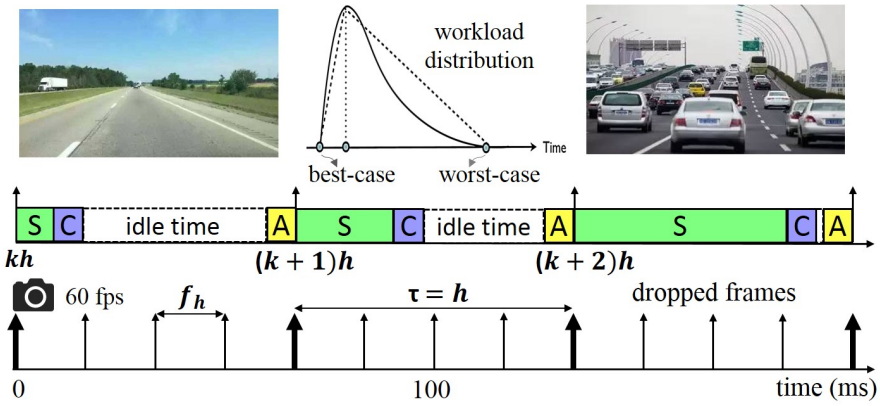



Figure 3.2: Illustration of a workload distribution and a sequential IBC implementation considering worst-case image workload. (Adapted from Fig. 1.4 (b) and (c), for readability).

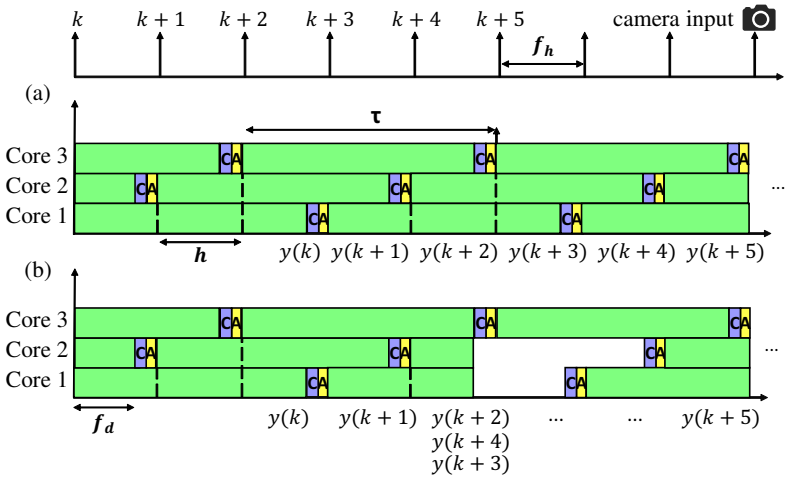



Figure 3.3: Illustration of pipelined IBC system implementation with constant sampling period $h$ and: (a) with constant worst-case sensing delay; (b) considering workload variations.

## 3.2 Pipelined IBC system implementation

We consider a typical setting for an IBC system as shown in Fig. 3.1 having the workload distribution as illustrated in Fig. 3.2. The main sensor is a camera module that captures the image stream. The image stream is then fed to an embedded multiprocessor platform at a fixed frame rate or frames per second (FPS), e.g., 60 fps. The image arrival period $f_h$ is the inverse of the frame rate, e.g., $f_h = 1/60 = 16.67$ ms. The tasks in such an application primarily include compute-intensive image sensing and processing (S), control computation (C) and actuation (A) which are then mapped to run on a multiprocessor platform.

In a pipelined control implementation (see Fig. 3.3), the sensing operations to read and process the system states start periodically at $t_s(S) = kh$, where $k$ is a non-negative integer. The sampling period $h$ is the interval between two consecutive activations of the sensing operation that require image frames for processing. We align $t_s(S) = kh$ with the availability of the frames as can be seen in Fig. 3.3 and hence, $h$ is an integer multiple of $f_h$.

Sensing and processing is followed by control computation and actuation operations, which generally take short and nearly constant time for execution. A sensing operation takes much longer time due to compute heavy processing, i.e., $e_S \gg e_C + e_A$, where $e_S$, $e_C$ and $e_A$ are the worst-case execution times of sensing and data processing, control computation and actuation tasks, respectively. The total (worst-case) execution time of a loop is thus given by $e_{\text{total}} = e_S + e_C + e_A$.

For a pipelined implementation, the sensor-to-actuator delay $\tau > h$ and it can

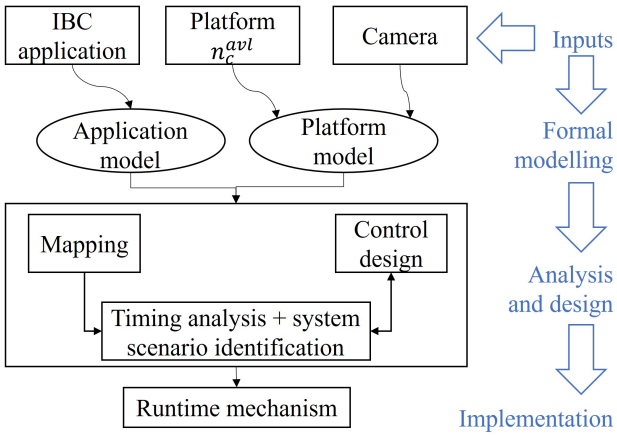



Figure 3.4: Basic SPADE flow for pipelined implementation without parallelising the sensing task. For the scope of this chapter, we assume that each pipe in the pipeline is mapped to a single (unique) processing core and the cores are homogeneous.

be represented as [8]

$$\tau = (n_f - 1) f_h + \tau_f, \text{ where } 0 < \tau_f \leq f_h, \; n_f = \left\lceil \frac{e_{\text{total}}}{f_h} \right\rceil. \tag{3.1}$$

The number of frames arriving within $\tau$ is $n_f$. An assumption we make, for the scope of this chapter, is that each pipe in the pipeline is implemented on one processing core.

## 3.3 SPADE for pipelining

In this section, we explain the SPADE approach for pipelining without parallelising the sensing task. The basic SPADE flow for the pipelined implementation is illustrated in Fig. 3.4. A key input parameter is the total number of available cores $n_c^{avl}$ that can be used. We assume for now that each pipe in the pipeline is mapped to a single (unique) processing core and that there is no resource sharing between the individual pipes. In addition, the allocated processing cores are assumed to be homogeneous so that mapping a single pipe to any of the cores would result in the same throughput and latency. These assumptions are relaxed in Chapter 4.

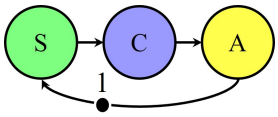



Figure 3.5: Scenario-aware dataflow (SADF) model of a single pipe for the generic pipelined implementation. Execution time of sensing task $S$ varies per scenario.

### 3.3.1 Formal modelling and mapping

We can model the application and the platform as explained in Section 2.3. Since there is no resource sharing between the individual pipes and each pipe is mapped to a unique processing core, we can use the generic SADF model shown in Fig. 3.5 to compute the sensor-to-actuator delay and sampling period for a single pipe. The scenario is determined by the execution time of the sensing task S. When there is resource sharing between the pipes, the SADF model needs to be transformed for mapping and scheduling. The model transformations are non-trivial and are explained later in Section 4.4.



### 3.3.2 Timing analysis and system scenario identification

The sensor-to-actuator delay for a single pipe can be computed for the generic SADF graph illustrated in Fig. 3.5 using the analysis method explained in Section 2.3.1. We compute the best-case and worst-case sensor-to-actuator delays for the best-case and worst-case workload scenarios per pipe. The best-case workload scenario results from the shortest execution time for the sensing task and the worst-case workload scenario results from longest execution time for the sensing task. After we compute the worst-case delay, we need to compute the effective sampling period $h$ considering the inter-frame dependencies and the platform constraints, explained below. For the adaptive MPC formulation-based controller design, we only need to do a runtime adaptation for considering workload variations (explained later in Section 3.5.2) and we do not need to reconfigure the mapping and scheduling (as was needed for the basic flow of the previous chapter). This means that for the implementation proposed in this chapter, there is only one system scenario. This system scenario should know the effective period $h$, worst-case delay $\tau$, effective number of frames skipped by a single pipe $n_d$ (defined below), and the number of processing cores needed to maximise the gains from pipelining $n_c^{max}$ (defined below). Because of inter-frame dependencies, there is a limit on the number of cores that gives performance benefits with pipelining. $\tau$ is initially computed by mapping the synchronous dataflow (SDF) graph of the worst-case workload scenario for a single pipe to one processing core of the given platform allocation. We can then compute the $n_c^{max}$ after considering the inter-frame dependencies. The effective period $h$ is then computed based on $n_c^{max}$ and the total number of available cores $n_c^{avl}$. Finally, we can compute $n_d$ such that the delay $\tau$ can be expressed in multiples of the sampling period as $\tau = n_d h + \tau'$, where the remainder $\tau'$ is $0 \leq \tau' < h$.

#### Adaptation with inter-frame dependencies

Inter-frame dependencies capture the data or algorithmic dependencies between consecutive frame processing. Considering inter-frame dependencies is crucial for practical real-life implementation. Inter-frame dependence time $f_d$ is the max-

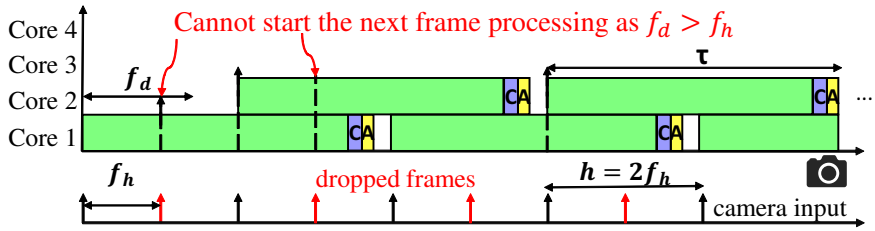



Figure 3.6: Illustration of inter-frame dependencies with $f_d > f_h$. Note: 1) Even if more cores are available they cannot be used due to inter-frame dependencies; 2) Our method as compared to [112] does not restrict $\tau$ to be an integer multiple of $f_h$; 3) If $f_d \leq f_h$ then $h = f_h$ using all four processing cores.



imum time required to wait between processing consecutive image frames due to inter-frame dependencies. Fig. 3.6 illustrates the impact of inter-frame dependence time on sampling period. In a pipelined implementation, considering inter-frame dependencies means that strictly $h \geq f_d$. Further, $h$ should be an integer multiple of $f_h$. For the scope of this chapter, we assume that $f_d$ is known or can be computed. The computation of $f_d$ is explained later in Chapter 4.

Inter-frame dependencies mean that sometimes some image frames have to be skipped for processing with respect to the given image arrival period $f_h$ and the sampling period $h$. Skipping a frame means that $h$ increases and thus degrades the control performance [112]. The number of frames that has to be skipped after processing every frame due to inter-frame dependencies is $n_s - 1$, as illustrated in Fig. 3.6 where $n_s = 2$ and one frame is skipped after every frame processing. The number of frames we have to skip $n_s - 1$ is determined by the inter-frame dependence time $f_d$. The earlier mentioned $n_d$ needed for defining the system scenarios is computed based on the effective $h$ and $\tau$ at runtime; we always have that $n_s \leq n_d$. $n_s$ is a constraint imposed by the inter-frame dependencies and $n_d$ is the effective number of frames we skip at runtime considering the platform constraints and $n_s$.

The effective image arrival period or the minimum possible sampling period $h_{min}$ we can then have is

$$h_{min} = n_s \times f_h, \text{ where } n_s = \left\lceil \frac{f_d}{f_h} \right\rceil.$$

The assumption here is that a sufficient number of processing cores $n_c^{avl}$ is available for pipelining.

#### Adaptation with the available number of processing cores

Another crucial aspect to consider for practical implementation is the number of available processing cores. A maximal pipelined implementation is defined as the pipelined implementation without skipping or dropping feasible image

frames considering inter-frame dependencies and camera frame rate. A maximal pipelined implementation is achieved when the realisable periodic sampling period $h = h_{min}$. E.g. Fig. 3.6 illustrates a maximal pipelined implementation with $n_s = 2$. The number of processing cores needed to realise the maximal pipelined implementation $n_c^{max}$ and the effective sampling period $h$ considering $n_c^{avl}$ are defined as follows:

$$n_c^{max} = \left\lceil \frac{n_f}{n_s} \right\rceil,\ h = \left\lceil \frac{n_c^{max}}{n_c^{avl}} n_s \right\rceil \times f_h, \text{ if } n_c^{avl} < n_c^{max},$$
$$= n_s \times f_h, \text{ otherwise.}$$



where $n_c^{avl}$ is the total number of available processing cores. Having more cores, i.e. $n_c^{avl} > n_c^{max}$, does not help as there are no more frames available for pipelining. However, if $n_c^{avl} < n_c^{max}$, there are not enough cores available to process the arriving frames $n_f$ and thus $h$ has to be increased and a maximal pipelined implementation cannot be achieved.

### 3.3.3 Controller design and workload variations in a pipelined IBC system

The contribution of this chapter is the control design method using an adaptive MPC formulation for the pipelined implementation. The controller design needs to know $h$, $\tau$ and $n_d$. The details are explained in Sections 3.4 and 3.5. In this section, we first analyse the impact of workload variations in a pipelined implementation.

The workload variations occur due to varying features in image content (see Fig. 3.2). When we do not consider workload variations, a pipelined implementation results in constant $\tau$ and $h$, as illustrated in Fig. 3.3(a). Notice that here we measure the outputs $y(k+i)$ for the input image frame at $k+i$ with a constant worst-case sensing delay of $\tau$, where for simplicity of notation, by $k+i$ we denote the time instant $(k+i)h$ with $i$ an integer.

Considering workload variations in a pipelined implementation of an IBC system implies that we would have varying sensing delays, e.g., as illustrated in Fig. 3.3(b). For this example, notice that the camera input frame at $k+4$ has a sensing delay of one frame ($\tau_1 = h$), at $k+3$ has a sensing delay of two frames ($\tau_2 = 2h$) and all other frames have a sensing delay of three frames ($\tau = 3h$). This scenario results in multiple sensing and image processing (S) tasks completing their execution at the same time. What this means is that multiple output measurements $y(k+2)$, $y(k+3)$ and $y(k+4)$ are available for control computation task $C$ at the same time instance and no measurements arrive at the next two sampling instances.

Thus, the main challenge for the pipelined IBC system design to maximize performance, i.e. QoC, is to effectively use the sensor measurements as early as possible for control computation without any unnecessary idling and to predict the

system state when there are no sensor measurements available. Modelling this behaviour is far from trivial.

### 3.3.4 Runtime mechanism

The mapping and scheduling is static at runtime for the pipelined implementation explained in this chapter. A runtime mechanism is needed to keep track of the latest sensing measurement that is available for the MPC computation. The sensing task should store the timestamp of the latest image processed and the corresponding sensing measurement in $p$ memory locations (one for each pipe). At the start of the control compute task, the runtime mechanism needs to read the $p$ memory locations with the timestamps, compute the $n_d$ for each and choose the sensing measurement corresponding to the lowest $n_d$ value. The MPC also requires the $n_d$ value to adapt the optimization problem formulation as explained in Section 3.5.2. The overhead for reading the $p$ memory locations and computing $n_d$, though negligible, is part of the control compute task.



## 3.4 Modelling and discretization

In this chapter, we consider a broad class of systems that can be described via LPV models for the predictive control approach. Specifically, this section first describes continuous-time state-space linear models which are typically obtained from first-principles. Next, a discretization scheme is described followed by details on transformation of the linear model to (I/O) form which is more suitable for the proposed control method considering the varying sensor-to-actuator delay.

### 3.4.1 Continuous-time model with input delay

The continuous-time LPV model we consider can mathematically be described at time $t$ as

$$\dot{x}(t) = A_c(p)x(t) + B_c(p)u(t-\tau) \tag{3.2a}$$

$$y(t) = C_c(p)x(t) \tag{3.2b}$$

where $x \in \mathbb{R}^{n_x}$ denotes the state vector, $y \in \mathbb{R}^{n_y}$ contains measured outputs and $u \in \mathbb{R}^{n_u}$ is the vector of control inputs. Vector $p \in \mathbb{R}^{n_p}$ contains the scheduling parameters which determine the model coefficients in matrices $A_c$, $B_c$ and $C_c$ as shown in Eq. (3.2). The sensor-to-actuator delay is denoted by $\tau$ and $\tau > 0$. Continuous-time models are useful for an accurate simulation of the system under study; however, for computer-based control, it is necessary to have a discrete-time model considering sampled signals.

### 3.4.2 Discrete-time model

**State-space description**

Based on a zero-order hold (ZOH) approximation where we assume the input signal to be constant over each sampling interval, we can use the methods described in [8] to obtain the discrete-time LPV model

$$x(kh+h) = \Phi(p)x(kh) + \Gamma_0(p,\tau')u(kh-n_dh) + \Gamma_1(p,\tau')u(kh-n_dh-h) \quad (3.3)$$



where $h$ is the sampling period, $k$ is an integer indicating the time step, and

$$\Phi(p) = e^{A_c(p)\cdot h} \quad (3.4a)$$

$$\Gamma_0(p,\tau') = \int_{\tau'}^{h} e^{A_c(p)\cdot(h-s)}\mathrm{d}s \quad (3.4b)$$

$$\Gamma_1(p,\tau') = \int_{0}^{\tau'} e^{A_c(p)\cdot(h-s)}\mathrm{d}s \quad (3.4c)$$

such that the total delay $\tau$ can be expressed in multiples of the sampling period as $\tau = n_dh + \tau'$, where the remainder $\tau'$ is $0 \le \tau' < h$. In practice, only a numerical approximation of the matrix exponential is used to compute the model matrices in Eq. (3.4), for which several methods exist. Specifically, for the example discussed in Section 3.6, we approximate the matrix exponential by using its 12th degree Taylor polynomial.

When some states do not need to be controlled, the size of the control problem may unnecessarily become large especially if the number of output variables is relatively small. This motivates the use of I/O models for control, which also allow an easy incorporation of delay as shown in Sections 3.4.2-3.4.2. In the linear model case, the I/O equations may simply be derived from the equivalent transfer function of the state-space model. Note also that linear models obtained from data-based system identification methods are often parameterized in I/O form.

**Input/output difference equations**

The state-space model (3.3) can be written as the following I/O model

$$y(k) = C_c(p)\left(q\boldsymbol{I} - \Phi(p)\right)^{-1}\Gamma_0(p,\tau')u(k-n_d) + C_c(p)\left(q\boldsymbol{I} - \Phi(p)\right)^{-1}\Gamma_1(p,\tau')u(k-n_d-1)$$

where $q$ denotes the forward shift operator such that $qy(k) = y(k+1)$ and $q^{-1}y(k) = y(k-1)$ and for ease of notation we dropped $h$ by assuming the time

scale in terms of sampling period. The symbol $\boldsymbol{I}$ denotes an identity matrix of appropriate size. On simplification of the above difference equations, the multivariable LPV model can be rewritten in the noise-free auto-regressive exogenous (ARX) form as

$$y(k) = \sum_{j=1}^{n_x} A_j(p)\, y(k-j) + \sum_{j=1}^{n_x+n_d+1} B_j(p,\tau')\, u(k-j) \tag{3.5}$$

where the coefficient matrices $A_j(p)$ are derived by evaluating the determinant of $\big(q\boldsymbol{I} - \Phi(p)\big)$ whereas entries of $B_j(p,\tau')$ are derived from its adjugate matrix, $C_c(p)$, $\Gamma_0$ and $\Gamma_1$.



#### Adapting the I/O model with time delay

Observing (3.5), it is clear that for $j = \{1,\ldots,n_d\}$, $B_j = \mathbf{0}$, where $\mathbf{0}$ is a zero matrix. Therefore, an increase in delay $n_d$ implies an according zeroing of the foremost input coefficients, without a change in the size of the MPC problem as clarified in Section 3.5. This also allows fixing the memory allocation for the model and the control algorithm based on the worst-case delay which can reasonably be assumed to be known. For further simplicity in the design and implementation of the controller, we assume $\tau'$ and $h$ to be constant. The delay $\tau'$ can be set to zero, especially when it varies, by a unit increment in $n_d$ to simplify the model evaluation as $\Gamma_1$ becomes a zero matrix referring Eq. (3.4c). The influence of this simplification is negligible when $h$ is sufficiently small.

## 3.5 Predictive control strategy

This section discusses the details on formulating the adaptive predictive control problem based on the LPV model (3.5) with varying time delay. The reader is referred to [20] for basic terminology and details related to MPC.

### 3.5.1 Optimization problem formulation

The MPC problem is formulated based on a performance index which reflects the control objectives and a set of constraints including the equalities due to the LPV prediction model (3.5). We consider a quadratic performance index $J$ that penalizes output tracking error and deviation of inputs from steady-state targets, i.e.,

$$J(k) = \sum_{j=1}^{N_\mathrm{p}(k)} \frac{1}{2} \| W_y(k+j) \cdot (y(k+j) - y_r(k+j)) \|_2^2 + \sum_{j=0}^{N_\mathrm{p}(k)-1} \frac{1}{2} \| W_u(k+j) \cdot (u(k+j) - u_r(k+j)) \|_2^2$$

where $W_y(k)$ and $W_u(k)$ denote weights on output and input respectively at time step $k$ whereas $y_r$ and $u_r$ denote their target values. The prediction horizon $N_\mathrm{p}(k)$ determines the number of decision variables i.e., the inputs and outputs in prediction to be optimized. Considering simple bounds on the decision variables, the MPC problem to be solved at each time step is the constrained optimization problem

$$\min_{u(\cdot),y(\cdot)} J(k)$$

$$\text{s.t.} \quad y(k+l) = \sum_{j=1}^{n_x} A_j(p)y(k+l-j) + \sum_{j=1}^{n_x+n_d(k)+1} B_j(p)u(k+l-j)$$

$$= M(p,n_d(k))\cdot\phi(k+l-1), \forall l \in \{1,\ldots,N_\mathrm{p}\} \tag{3.6a}$$

$$u(k+j) = u(k+N_\mathrm{u}-1), \forall j \in \{N_\mathrm{u},\ldots,N_\mathrm{p}\} \tag{3.6b}$$

$$y(k) = y_0 \tag{3.6c}$$

$$\phi(k) = \phi_0 \tag{3.6d}$$

$$y_\mathrm{l}(k+j) \le y(k+j) \le y_\mathrm{u}(k+j), \forall j \in \{1,\ldots,N_\mathrm{p}\} \tag{3.6e}$$

$$u_\mathrm{l}(k+j) \le u(k+j) \le u_\mathrm{u}(k+j), \forall j \in \{1,\ldots,N_\mathrm{u}\} \tag{3.6f}$$

where $M$ in (3.6a) is the matrix of parameter-dependent model coefficients such that $\phi(k) = [y^\top(k), y^\top(k-1),\ldots,y^\top(k-n_x), u^\top(k-1), u^\top(k-2),\ldots,u^\top(k-n_x-n_d(k)-1)]^\top$. The upper and lower bounds on any variable $z$ are denoted as $z_\mathrm{u}$ and $z_\mathrm{l}$, respectively. The initial condition (3.6c)-(3.6d) in the I/O case is provided via the measured output feedback $y_0$, and vector $\phi_0$ which also includes the known past sequence of inputs and outputs. The input sequence that is optimized is typically restricted to fewer variables for trading-off control performance with computations via the control horizon ($N_\mathrm{u}$) constraint (3.6b), i.e., by tuning the parameter $N_\mathrm{u}$ such that $1 \le N_\mathrm{u} < N_\mathrm{p}$, where $N_\mathrm{p} > n_d$. Note that the future values of parameters $p$ can be incorporated as they may be known and in that case $p$ represents $p(k+l)$ in the LPV model (3.6a).

### 3.5.2 Adaptation with workload variations

In IBC, besides control computation and actuation time, the sensor-to-actuator delay mainly includes the sensing time. We assume that the delay due to control computation and actuation are fixed, but the sensing delay may vary as explained in Section 3.3.3. We propose to adapt the controller at runtime to make immediate use of the latest available measurement in order to maximize QoC. The basic idea is to have the time delay as a variable parameter based on which the model is adapted as explained in Section 3.4.2. The varying parameter ($n_d$) is then kept constant in prediction as shown in (3.6a). Since the actuation rate can be constant thanks to pipelining, a new control action is computed at each time step with a ZOH during the sampling period.

The following three cases may occur at each time step due to varying sensing delay (illustrated in Fig. 3.7):

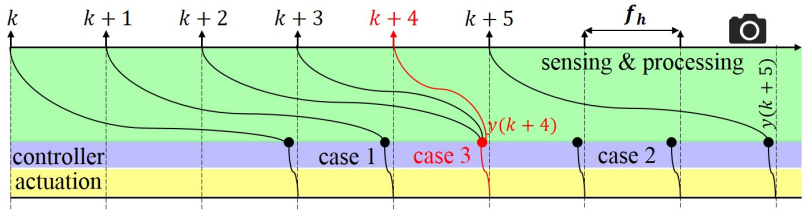



Figure 3.7: Illustration of cases described in Section 3.5.2. The samples $k+3$ and $k+4$ have lower workloads and thus the latest output measurement $y(k+4)$ is available within one $f_h$. Note that for case 2, the latest measurements are not available and thus $u(k+i)$ is computed based on the MPC prediction model.



1. the new measurement is available with the same sensing delay as in the previous step: in this case the parameter $n_d$ is kept constant and the initial condition for computing the new control input is updated as usual, i.e., $\phi = [y^\top(k-1),\dots,y^\top(k-n_x),\cdots]^\top$ becomes $\phi = [y(k),\dots,y^\top(k-n_x+1),\cdots]^\top$;

2. the sensing delay is increased compared to the previous step as the latest measurement is not available: in this case, the prediction is done starting from the old measurement, i.e., the delay parameter $n_d$ is incremented by 1 and the stack of past outputs in $\phi$ is not updated.

3. the sensing delay is reduced by one or more steps: when multiple pipes finish processing a corresponding sequence of frames, both the latest measurement(s) along with the past measurements now available are fed to the controller and $n_d$ is accordingly reduced where, from an implementation perspective, the output stack in the initial condition $\phi$ is updated as in case 1 discussed above and the same procedure is then repeated as many times as the reduction in $n_d$ before computing the next control input.

Since a new control input is applied at each step, the stack of past inputs in the initial condition vector $\phi$ is always updated by a unit shift in all the three cases, i.e., in $\phi = [\cdots,u^\top(k-1),\dots,u^\top(k-n)]^\top$ becomes $\phi = [\cdots,u^\top(k),\dots,u^\top(k-n+1)]^\top$. In conclusion, until the next measurement is available, the proposed controller compensates for the delay by making use of the prediction model to implicitly estimate the current status of the system (without a separate open-loop estimator) while accordingly computing an optimal action that satisfies given constraints.

The ordering of measurements is important in the current MPC implementation since we do not want to allow discarding measurements (as the application we consider, in Section 1.7, is safety-critical). This can be considered as a limitation of the current approach. Ordering means that the latest measurement $y(k+i)$ is updated in the output stack $\phi$ iff the previous measurements $y(k+j)$, $j<i$, were already updated in $\phi$.

## 3.6 Simulation results and comparison

We illustrate our work using the motivating case study of a vision-based lateral control system explained in Section 1.7 and described as follows.

$$\dot{x}(t) = A_c(v_x)x(t) + B_c u(t-\tau),\ y(t) = C_c x(t), \tag{3.7}$$

### 3.6.1 MPC implementation



Based on a discretized version[1] of (3.7) using methods discussed in Section 3.4.2, the optimization problem (3.6) can be formulated for MPC with longitudinal velocity $v_x$ and delay parameter $n_d$ as the scheduling variables. Constraints are imposed on the control input $\delta_f$. In (3.6a), $v_x$ may be assumed either to be a constant in prediction or a variable, as described in Section 3.5.1. Since for the considered lateral control system, problem (3.6) has a quadratic cost function subject to linear constraints, it can be solved using any quadratic programming (QP) algorithm. We use the methods described in [115] for an efficient implementation. Specifically, the QP problem (3.6) is transformed to the following box-constrained least-squares problem by eliminating equality constraints through quadratic penalties with large weight $\rho$, i.e.,

$$\min_{u(\cdot),y(\cdot)} J(k) + \rho \sum_{l=1}^{N_\mathrm{p}} \left\| y(k+l) - M(p, n_d(k)) \cdot \phi(k+l-1) \right\|^2 \tag{3.8}$$

$$\text{s.t. (3.6e)-(3.6f)}, \tag{3.9}$$

and by substituting (3.6b)-(3.6d) in the cost function. Following [115], we implement the optimization algorithm such that it is not only able to automatically adapt to changes in parameters $p$ including $n_d$ but also MPC parameters such as the horizons $N_\mathrm{u}$, $N_\mathrm{p}$, and the tuning weights through which the controller can be re-tuned at run time. Besides this, as problem (3.8) is always solvable, this method has a practical benefit as it is also able to deal with situations under which the constrained QP (3.6) might become infeasible to solve due to model mismatch and unmeasured disturbances.

### 3.6.2 Controller performance evaluation

This section includes simulation results for our case study. We consider two scenarios: 1) the adaptive MPC algorithm is run with a constant worst-case delay, i.e. neglecting workload variations, and 2) with delay as a variable to explicitly consider workload variations, without changing any tuning parameters. The purpose of this simulation is to highlight the benefits of the control design that also adapts

[1] The symbolic math toolbox of MATLAB R2015b was used for obtaining the required discrete-time LPV model from (3.7)

Table 3.1: Comparison between the proposed pipelined SPADE MPC approach with the state-of-the-art multiprocessor IBC system implementations

| Criteria | SPADE pipelined MPC | Pipelined | | Chapter 2 |
|---|---|---|---|---|
| | | constant delay [112], [69] | variable delay [82] | |
| Inter-frame dependencies | explicitly considered | not considered | not considered | independent |
| System nonlinearities | explicitly considered | not considered | not considered | not considered |
| Constraints on variables | can be strictly imposed | cannot be imposed | cannot be imposed | cannot be imposed |
| Control computation time | high (worst-case up to 15x greater than Chapter 2) | low | medium (a delay predictor needed) | low (feedback gain matrix multiplication) |
| Algorithm | white/gray/black box | white/gray/black box | white/gray/black box | white/gray box |
| Parallelisation potential | independent | independent | independent | should be high |
| Workload variations | explicitly considered in design | not considered | indirectly considered as variable delay | explicitly considered in design |
| Platform | suitable for homogeneous | suitable for homogeneous | can be adapted for all | directly applicable for all |
| Restrictions on $h$[1] | strictly periodic; $h < \tau_{wc}$ | strictly periodic; $h < \tau$ | strictly periodic; $h < \tau_{wc}$ | switched system possible |
| Restrictions on $\tau$ | strictly[2] $\tau_{wc} > h$ | strictly $\tau > h$; in [112], $\tau$ is strictly a multiple of $h$ | strictly[2] $\tau_{wc} > h$ | $\tau_{wc} \leq h$ |

$\tau_{wc}$: worst-case delay; [1] If camera frame arrival period $f_h$ is considered, always $h$ is a multiple of $f_h$; [2] if $\tau \leq h$ design reverts to sequential;

well with workload variations. The influence of delay is apparent with model mismatch and unmeasured disturbances. Since the true system considered is the continuous-time model (3.7) and discretization errors are negligible, in order to emulate the influence of realistic model mismatch and unmeasured disturbances, we provide the output reference for vehicle lateral control along with the output measurement, i.e., with a varying delay. Note that this is done only for this particular simulation scenario and is not the case in practice where the reference is already known.

We assume the camera frame rate of 60 fps, i.e., $f_h = 1/60$ s. Simulation length is 5000 time steps i.e. $T = 83.33$ s. Unit weights are imposed on all I/O variables for MPC, while $N_\mathrm{p} = N_\mathrm{u} = 10$ time steps. The steady-state input reference $u_r(t) = 0$ whereas the output reference profile is set to a sinusoidal signal such that at time step $k$, $y_r(k) = 2.5 \cdot \sin(5kh\pi/T)$ m. The longitudinal velocity is a ramp signal such that $v_x(0) = 45$ and $v_x(T) = 80$ km/h, which we assume to be known in prediction for the controller.

The mean runtime for the control algorithm was 2.2 ms while solving the optimization problems with 20 decision variables in MATLAB (on a computer equipped with a 2.6GHz processor). Referring to the reported runtimes in [115] for control problems of comparable size, we expect a similar efficient embedded implementation of the algorithms with a C backend to significantly reduce these runtimes (roughly by 20x, to about 0.1 ms) on a processor with comparable specifications. Assuming that the target platform is around 60 times slower, the worst-

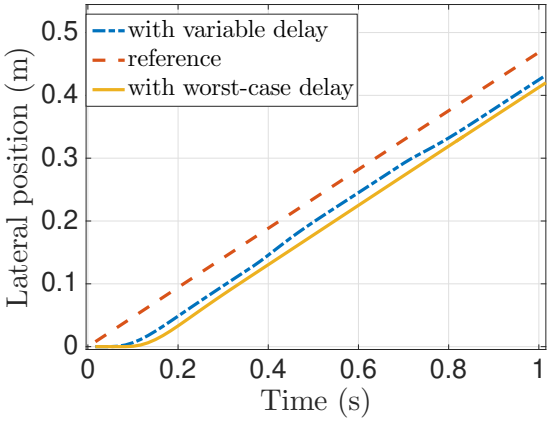



Figure 3.8: Lateral position of the vehicle w.r.t. the road centre.

case execution time (WCET) of tasks S, C and A are $e_S$= 60 ms (based on Chapter 2), $e_C$= 6 ms [115] and $e_A$= 0.5 ms, respectively. This results in worst-case delay $\tau$ = 66.5 ms and $n_f$ = 4.

We assume that the inter-frame dependence time $f_d$ is given and $f_d$ = 15 ms. So $n_s$ = 1 and the maximum number of cores needed $n_c^{max}$ = 4. We assume that $n_c^{avl}$ = 4 and thus $h$ = 1/60 s. To simulate the workload variations for the variable delay scenario, we consider the delay to be a random signal where the delay may take any value $n_d h$ which is held constant for every 20 frames such that $n_d \in \{1,2,3,4\}$ with the corresponding probability distribution of occurrence as {0.2435,0.3534,0.3073,0.0958}. Such a probability distribution for characterising workload variations can be statistically analysed from observed data [46].

Based on the aforementioned simulation implemented in MATLAB on a computer equipped with a 2.6GHz processor, the results obtained show that a root mean square error (RMSE) of 2.98 cm from the output reference is achieved using MPC considering a variable delay. For MPC based on constant (worst-case) delay the RMSE increases by 26.85% to 3.78 cm, as is clearly seen in Fig. 3.8. The improvement by handling workload variations is expected to be higher when the worst-case delay is considerably larger than its mean.

In the simulation, we considered the following aspects which are crucial for real-life practical implementation. 1) inter-frame dependencies: $f_d$ = 15 ms; 2) system nonlinearities: $v_x$ is a ramp signal; 3) constraints: $\delta_f$ constrained to maximum magnitude of 0.5 radians; and 4) workload variations as a variable delay based on probability distribution.

### 3.6.3 Comparison with the state-of-the-art

We compare the proposed MPC formulation for pipelined IBC implementation with state-of-the-art multiprocessor IBC design techniques in Table 3.1. For

brevity, we only compare with multiprocessor IBC system implementations and not with traditional sequential control design techniques based on the worst-case sensing delay, as it has already been shown in [46] that considering workload variations is beneficial for optimizing control performance. The multiprocessor implementations can be classified into pipelined [112], [69] with constant delay, pipelined with variable delay [82] and sequential implementation with parallelisable sensing (explained in Chapter 2). The camera frame rate, however, is not explicitly considered in [69].

The proposed approach is advantageous to others with respect to: 1) considering inter-frame dependencies; 2) modelling and considering system nonlinearities; and 3) strictly imposing constraints on the system variables. These aspects are crucial for practical implementation and explicitly considering them helps in making a step forward towards real-life adaptation. The proposed approach, however, requires higher worst-case control computation time $e_C$ (up to 15x greater than in Chapter 2) due to solving the online optimization problem. Note that $e_C$ is not yet significant compared to the sensing workload, i.e., $e_S \gg e_C + e_A$. Future work includes identifying a case where $e_C$ can be significant compared to the sensing workload and adapting our method for it.



## 3.7 Conclusions

We presented a pipelined, multiprocessor, adaptive MPC formulation for IBC systems while considering workload variations, inter-frame dependencies, system nonlinearities and constraints on system variables. The proposed approach aims to reduce the gap between current state-of-the-art multiprocessor IBC approaches and practical control requirements while optimizing quality-of-control and making a step forward towards real-life adoption. First results based on simulations suggest that by using the proposed method one can address practical implementation challenges not directly dealt with in the past approaches, which either did not consider variations in sensing delay, or inter-frame dependencies, presence of nonlinearities in system dynamics, or practical constraints on system variables. The results presented in this chapter assume that each pipe in the pipeline is mapped to a single (unique) processing core and there is no resource sharing between the individual pipes. In addition, the allocated processing cores are assumed to be homogeneous so that mapping a single pipe to any of the cores would result in the same throughput and latency. These assumptions will be relaxed in Chapter 4.

# 4

# SPADE by pipelined parallelism



Image-based control (IBC) systems have a long sensing delay due to compute-intensive image processing. Modern multiprocessor IBC implementations consider either parallelisation of the sensing task or pipelining of the control loop to cope with this long delay. Chapter 2 already discusses multiprocessor IBC implementations considering parallelisation of the sensing task, and Chapter 3 discusses the pipelining of the control loop without parallelising the sensing task. The impact of both parallelisation and pipelining together on the quality-of-control (QOC) of IBC systems was not explored in the literature prior to this work. We present the complete version of the scenario- and platform-aware design (SPADE) approach (briefly summarised in Chapter 1) for multiprocessor IBC implementation, considering both parallelisation and pipelining together. In particular, we address the following problem: For a given platform allocation, what is the optimal degree of pipelining and degree of parallelisation required to maximise the QOC? The proposed method takes into account image-workload variations, inter-frame dependencies and platform constraints. The application is efficiently modelled and analysed using a scenario-aware dataflow graph, and an implementation-aware switched controller is designed that optimises QOC and guarantees stability. We validate the proposed method using simulations and hardware-in-the-loop experiments, considering the lane-keeping assist system (LKAS) already introduced in Chapter 1.

## 4.1 Background and contributions

IBC systems are feedback control systems whose feedback is provided by camera(s) as the sensor(s) (illustrated in Fig. 4.1 (a)). A camera captures image frames at a pre-defined constant frame rate from the dynamic system environment. A compute-intensive image-processing algorithm processes the image frames to detect features in the image such as objects, traffic signs and lanes. These features are then used to compute the states of the system, such as relative position and distance [29]. A controller computes the control input for actuation using the com-

The content of this chapter is an adaptation of the following paper:
Sajid Mohamed, Dip Goswami, Sayandip De, and Twan Basten. Optimising multiprocessor image-based control through pipelining and parallelism. *IEEE Access*, 9:112332–112358, 2021.

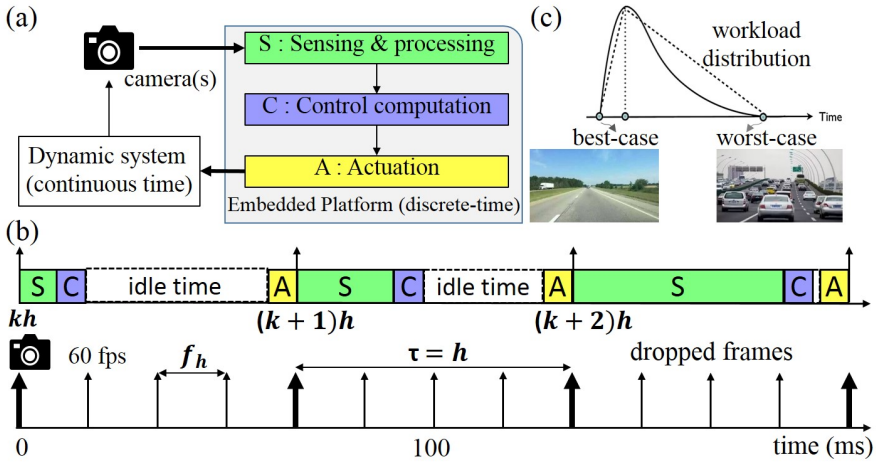



Figure 4.1: An IBC system: (a) block diagram; (b) Gantt chart for a typical IBC implementation; (c) workload variations captured as a distribution. (repeating Fig. 1.4, for readability)

puted states. The actuation task applies the computed control input to the IBC system.

A typical periodic implementation of such an IBC system is illustrated in Fig. 4.1 (b). The main challenge here is to deal with the inherent long (worst-case) sensing delay due to compute-intensive image-processing algorithms. A long processing delay results in dropping some camera frames from processing. Moreover, the sensing delay is variable due to image-workload variations [86]. These variations can be captured statistically using a probability distribution [1] (illustrated in Fig. 4.1 (c)). A long worst-case sensing delay leads to a long sensor-to-actuator delay $\tau$ (the time between the start of a sensing task and the end of the corresponding actuation task) and thus results in degraded control performance [8, 117]. The question is: *How to cope with the long variable sensing delay in an* IBC *system?*

The advent of multiprocessor platforms enables **coping with the long sensing delay** by either *parallelising the sensing task* (explained in Chapter 2) or *pipelining the control loop* [69, 112]. Parallelisation refers to executing sensing subtasks in parallel and thereby reduces the delay compared to the worst case (illustrated in Fig. 4.2 (a)). It is, however, limited by the degree of parallelism of the sensing algorithm. Pipelining refers to the pipelined execution of the control loop over multiple processing cores. Pipelining helps to reduce the number of camera frames being skipped. It reduces the sampling period $h$ (the time between the start of two successive sensing tasks) by processing frames on available cores (illustrated in Fig. 4.2 (b)). Pipelining is, however, limited by the presence of inter-frame dependencies, i.e., the data or algorithmic dependencies between consecutive frame

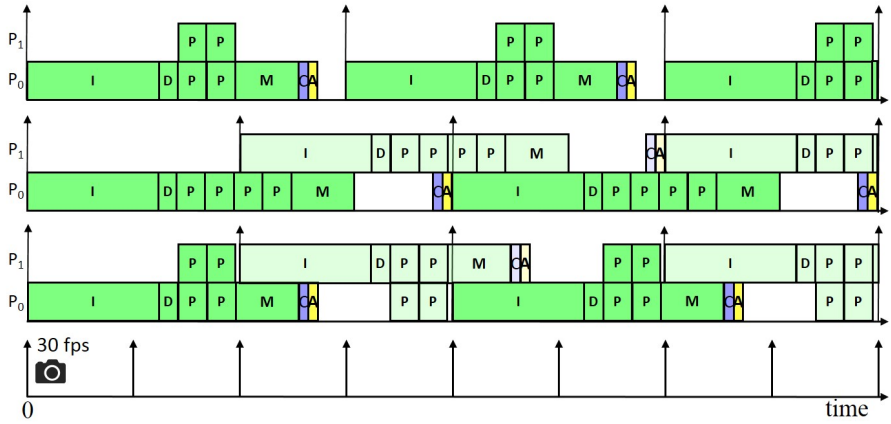



Figure 4.2: IBC implementations for worst-case image workload: (a) Parallelisation of sensing; (b) Pipelining without resource sharing; (c) Pipelining and parallelism together with resource sharing. Note: 1) Sensing task S is composed of image signal (pre-)processing (I), region-of-interest (ROI) detection D, ROI processing P, and ROI merging Mexplained in Section 4.3.1; 2) $P_0$ and $P_1$ are the two processing cores.

processing, e.g., due to video coding [77] or visual tracking [120]. In literature, the controller implemented for the pipelining is for $\tau > h$ [69, 112] and for parallelisation is for $\tau \leq h$ [91, 93] (and Chapter 2 of this thesis).

**Why should we consider pipelining and parallelism together?** Pipelining does not reduce the delay $\tau$ compared to the worst case. By executing the frames in a pipeline, only $h$ is reduced, whereas parallelising the sensing tasks reduces $\tau$. However, $h$ is still at least $\tau$ for a parallel implementation. Considering both pipelining and parallelism together helps to reduce both $\tau$ and $h$ and thereby improves the QoC of our IBC system. Further, the inherent limitations of pipelining - due to inter-frame dependencies - and parallelism - due to a limited degree of parallelism of the algorithm - can be mitigated significantly by considering them both together. The challenge then is to identify the optimal implementation choice considering both the degrees of pipelining and parallelism that improves the system performance. The degree of pipelining is quantified by the maximum number of active pipes in the pipeline, and the degree of application parallelism is quantified by the maximum amount of parallel execution within one sensing task. Both are limited by the available processing resources.

**Challenges:** The literature prior to this work does not explore the impact of both pipelining and parallelism together on the QoC of IBC implementation. Existing pipelined IBC implementations [69, 112] assume that the mapping is given, and that each pipe is mapped to a unique resource *without any resource sharing between pipes*. This is a restrictive implementation choice. Inter-frame depen-

dencies, which are crucial for practical implementation, are also not considered. We considered inter-frame dependencies in the previous chapter, and consider them integrally in combination with parallelisation in this chapter. There are two main challenges that were not explicitly explored prior to this work. First, how to model a multiprocessor IBC system considering both pipelining and parallelism together? The challenge in modelling is to explicitly consider workload variations, inter-frame dependencies and constant (often periodic) control timing parameters $\tau$ and $h$. Second, how to identify the optimal mapping of the sensing task on shared processing resources that considers both pipelining and parallelism together and provides a tight analytical bound on control timing parameters $\tau$ and $h$ so as to optimise QoC.



As already motivated, we chose scenario-aware dataflow (SADF) [129] as our model-of-computation (MoC) as it inherently supports modelling scenarios and has tool support for timing analysis and platform-aware mapping. Existing mapping analysis tools [5,123] typically assume that each node (subtask or actor) in the graph is bound to one processing resource. Pipelining involves (possibly) concurrent executions of subtasks on multiple resources, with inter-frame dependencies between actor instances. Moreover, control assumes careful time-triggered execution of sensing and actuation tasks. All these aspects can only be analysed after non-trivial graph transformations (as e.g. exemplified in [70]).

Fig. 4.2 (c) illustrates an implementation of two pipes on a shared platform allocation of two processors, with each pipe having a parallelised sensing subtask. With parallelised pipes but without resource sharing between pipes, we would need four processors to achieve the same delay and period as obtained in Fig. 4.2 (c). To integrally consider pipelining and parallelisation on a shared multiprocessor platform, we need an efficient analysis to identify the optimal mapping. The mapping should guarantee the required (often constant) worst-case delay and period for the controller design.

**The contributions** of this chapter are as follows:

1. We extend the SPADE approach, presented in Chapters 2, by considering pipelining of the control loop and formalising the IBC system modelling. This complete version of the SPADE approach (as explained in Section 4.3) integrally considers pipelining and parallelism for a multiprocessor IBC implementation. The exact problem addressed is the following: *For a given multiprocessor platform allocation, identify the optimal design choice for an* IBC *system considering both pipelining and parallelism and explicitly considering image-workload variations, inter-frame dependencies, resource sharing between pipes and platform constraints.* The optimal design choice identifies the degree of pipelining and degree of parallelism required for maximising the QoC and is translated into *system configurations* that guarantee control timing parameters.

2. We propose model transformations for modelling, analysing, and mapping

the IBC system. The model transformations for pipelined parallelism are the main contribution. These transformations consider both pipelining and parallelism together (as explained in Section 4.4). The model transformations allow us to relate the dataflow timing (throughput and latency) analysis to the key control timing parameters ($h$ and $\tau$) and to optimise the mapping while integrally considering pipelining and parallelism along with workload variations, inter-frame dependencies and resource-sharing between pipes. Implementation-aware model transformations for model-based design of IBC systems are not considered in prior literature.

3. We validate our approach using Matlab simulations considering a predictable multiprocessor platform - composable and predictable multiprocessor system-on-chip (CompSOC) [55] - and using hardware-in-the-loop (HiL) experiments with an industrial heterogeneous multiprocessor platform - NVIDIA AGX Xavier - considering a LKAS. Both platforms and the LKAS case study were already introduced in Chapter 1.



The rest of the chapter is organised as follows. Section 4.2 describes the multiprocessor IBC system implementation and the QoC metrics. Section 4.3 details the SPADe design flow. Section 4.4 introduces the model transformations required for the SPADe design flow to analyse pipelined parallelism. Section 4.5 revisits the SPADe design flow and precisely describes an algorithm using the model transformations and other considerations for pipelined parallelism. Section 4.6 explores the experimental results, the design-space exploration (DSE), and compares SPADe with the state-of-the-art multiprocessor IBC system implementations. Section 4.7 presents the SPADe adaptation for an industrial platform, the NVIDIA AGX Xavier, and validates the results of our approach in a HiL setting. Section 4.8 concludes the work and suggests possible future directions.

## 4.2 Multiprocessor IBC implementation

We consider a typical setting for an IBC system as shown in Fig. 4.1 (a) having the workload distribution as illustrated in Fig. 4.1 (c). The main sensor is a camera module that captures the image stream. The image stream is then fed to an embedded multiprocessor platform at a fixed frames per second (FPS), e.g. 60 fps and image arrival period $f_h$ given by $f_h = 1/60 = 16.67$ ms. The tasks include compute-intensive image sensing and processing (S), control computation (C), and actuation (A), which are then mapped to run on a multiprocessor platform. We illustrate our work using the motivating case study of a vision-based lateral control system model explained in Section 1.7. Further, we consider predictable and composable multiprocessor system-on-chip (MPSoC) platform CompSOC (explained in Section 1.3.1) for illustrating the SPADe flow for pipelined parallelism. We also adapt the SPADe approach for the industrial platform NVIDIA AGX Xavier (explained in Section 1.3.3) to demonstrate its applicability in an industrial context.

### 4.2.1 Control system and embedded implementation

We consider a linear time-invariant (LTI) feedback control system model for IBC given by:

$$\dot{x}(t) = A_c x(t) + B_c u(t), \qquad (4.1)$$
$$y(t) = C_c x(t) + D_c u(t),$$

where $x(t) \in \mathbb{R}^n$ represents the *state* vector, $y(t) \in \mathbb{R}$ contains the measured *output* and $u(t) \in \mathbb{R}$ represents the control *input* of the system at any time $t \in \mathbb{R}_{\geq 0}$. $A_c$, $B_c$, $C_c$ and $D_c$ represent the system, input, output and feedforward matrices of appropriate dimensions.

The model of Eq. 4.1 is a slight generalization of the model used in Chapter 2, including the feedforward matrix. The embedded implementation we consider is already explained in Section 2.2.2. We assume that the start of sensor-data processing is aligned with the camera frame arrival, i.e., $h$ is an integer multiple of $f_h$. Also, the control computation task and actuation task are delayed, if needed, to guarantee a constant $\tau$ such that $t_s(C^k) = t_s(S^k) + \tau - e_C - e_A$, and $t_s(A^k) = t_s(C^k) + e_C$. For a non-pipelined implementation (see Fig. 4.2 (a)), $\tau \leq h$, i.e., $t_s(S^{k+1}) \geq t_s(S^k) + \tau$. For a pipelined implementation (see Fig. 4.2 (b), (c)), $\tau > h$, i.e. $t_s(S^{k+1}) < t_s(S^k) + \tau$. With sensor-to-actuator delay $\tau$ and a zero-order-hold mechanism with sampling period $h \in \mathbb{R}$, $u(t)$ becomes piecewise constant in the intervals $t \in [kh + \tau,\ (k+1)h + \tau]$ for $k \in \mathbb{Z}_{\geq 0}$.

The main challenge here is to compute tight $\tau$ and $h$ for a multiprocessor IBC implementation. Identifying the optimal mapping that guarantees a constant $\tau$ and $h$, considering both pipelining and parallelism together, is non-trivial.

### 4.2.2 QoC metrics for control stability and performance

We evaluate the QoC of our IBC system design choices by considering stability and performance. Stability margins - gain and phase margins [125] - quantify the control stability and give an analytical basis to compare two different IBC system design choices and thus allow us to identify the optimal degree of pipelining and application parallelism. Once we make a choice, we can further optimise the controller with respect to performance, considering mean square error (MSE) (explained in Section 2.2.5) and settling time (ST).

The gain margin and phase margin quantify the additional gain and phase lag that makes the system marginally stable. Systems with greater stability margins can withstand greater changes in system parameters before becoming unstable. Gain margin and phase margin are computed analytically from the system model. On the other hand, MSE and ST can be analysed only through simulations.

**Gain margin (GM):** The gain margin (GM) is defined as the change in open-loop gain expressed in decibels (dB), required at 180 degrees of phase shift to make the system unstable. The GM is the difference between the magnitude curve and 0dB

at the point corresponding to the frequency that gives us a phase of -180 degrees (the phase cross-over frequency).
**Phase margin (PM):** The GM is the change in open-loop phase shift required at unity gain to make a closed-loop system unstable. The GM is the difference in phase between the phase curve and -180 degrees at the point corresponding to the frequency that gives us a gain of 0dB (the gain cross-over frequency).

The control performance quantifies, in essence, how fast the output $y(t)$ reaches the reference $r_{ref}$. The control performance can be tuned in the cost function for the control gains' design using the state and input weights [86].
**Settling time (ST):** The settling time is defined as the time required for the output $y(t)$ to reach and stay within a range of a certain percentage (usually 5% or 2%) of the final (reference) value $r_{ref}$ forever without external disturbances.



## 4.3 SPADE flow

We present the SPADE for IBC systems extending the approach presented in Chapter 2 by considering pipelining along with parallelism and formalising the IBC system modelling. An overview of our SPADE approach is illustrated in Fig. 4.3 (repeating Fig. 1.8, for readability), summarised below and explained in detail in subsequent subsections.

1. Formal modelling of the IBC system: An IBC application is captured as an IBC SADF model considering workload variations $W$ and the platform as a platform graph. Further, an *implementation-aware* IBC SADF model captures the given design parameters - camera frame arrival period $f_h$, maximum number of allowed pipes $p$, total number of available cores $n_c^{avl}$ and allocated processing cores for parallel execution per pipe $n_c^{//}$. The design parameters fully determine the implementation choice - non-pipelined without parallelism, non-pipelined with parallelism, pipelined without parallelism and pipelined with parallelism. The parallelism here refers to the parallel execution of sensing subtasks limited by the degree of parallelism of the IBC application. Graph transformations are proposed to obtain the implementation-aware SADF model.

2. Analysis and design: We map the implementation-aware IBC graph for each workload $s_i \in W$ to the platform graph to obtain the *binding-aware graph* $\mathcal{G}_i^b$ for that specific workload using the SDF3 mapping flow [122]. $\mathcal{G}_i^b$ is a synchronous dataflow graph (SDFG) that models the mapping of the implementation-aware graph to the platform graph. The mapping binds each actor in the SDFG to a processing core in the platform graph. For the ordering of execution of actors bound to the same core, a static-order schedule is encoded in the SDFG. A throughput and latency analysis of $\mathcal{G}_i^b$ yields the sensor-to-actuator delay $\tau_i$, and sampling period $h_i$. For a

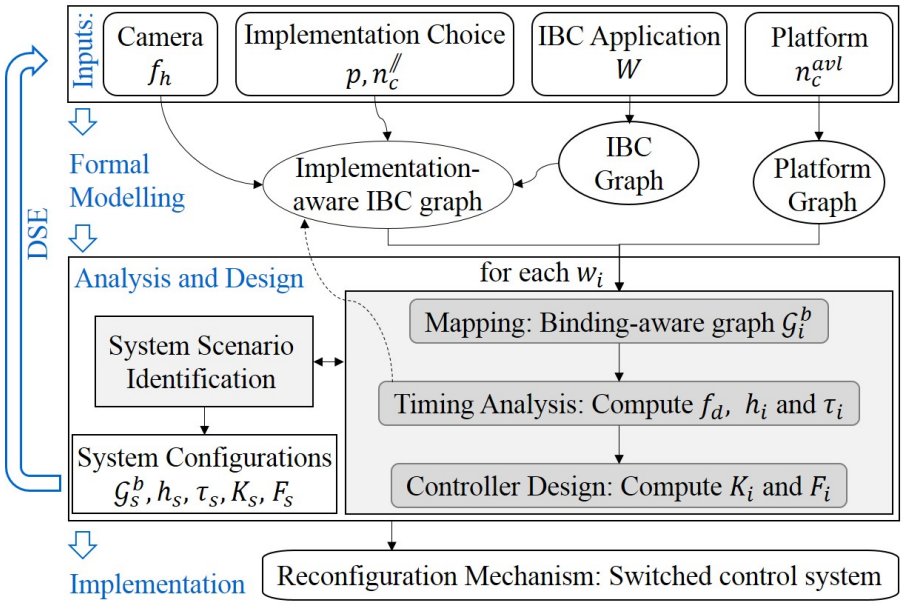



Figure 4.3: Overview of our SPADE design flow (repeating Fig. 1.8, for readability). $W$ is the set of varying workloads and $s_i$, $\mathcal{G}_i^b$, $\tau_i$, $h_i$, $K_i$ and $F_i$ are the workload, binding-aware graph, sensor-to-actuator delay, sampling period, feedback gain and feedforward gain for a workload scenario $s_i$ (determined by $s_i \in W$); $\mathcal{G}_s^b$, $\tau_s$, $h_s$, $K_s$ and $F_s$ are the corresponding parameters for an identified system scenario $s_s$ (that abstract multiple workload scenarios). $f_h$ is the camera frame arrival period, $f_d$ is the inter-frame dependence time, $p$ is the number of pipes for pipelining, $n_c^{//}$ is the number of cores allocated for parallelism per pipe, and $n_c^{avl}$ is the total number of available cores.

pipelined implementation, the throughput analysis of the worst-case image-workload scenario allows to compute the inter-frame dependence time $f_d$ (as explained later in Section 4.5.4). If $f_d > h_i$, the implementation-aware graph is updated with the realisable period and $\tau_i$ and $h_i$ are recomputed. The controllers are then designed for the resulting $(\tau_i,\ h_i)$ to obtain the controller feedback and feedforward gains $(K_i,\ F_i)$. Trying to cater to the designed workload scenarios at runtime means that we have a switching system. A switching system with too many switching states is challenging for controller stability and may result in poor performance. Hence, we aggregate multiple workload scenarios with similar control timing parameters as a *system scenario*. A system scenario $s_s$ abstracts multiple workload scenarios and has a constant $(\tau_s,\ h_s)$ during implementation. A *system configuration*

is defined as the combination of mapping and controller configurations, i.e. $\mathcal{G}_s^b$, $\tau_s$, $h_s$, $K_s$, and $F_s$ (as explained later in Section 4.3.2). Typically, there are a few identified system scenarios, and the idea is that switching between the system scenarios at runtime guarantees stability and improved performance. For pipelined parallelism, a DSE using the SPADE flow needs to be performed by varying the design parameters to identify the best implementation choice (parameters $p, n_c^{//}$, further explained in Section 4.6.1).

3. Runtime implementation: The system configurations for the implementation choice are stored in a look-up table (LUT) in platform memory for the runtime implementation. Dynamic runtime reconfiguration may be needed since there can be a switching behaviour between system configurations due to image-workload variations.



### 4.3.1 Formal modelling

An IBC application is captured as an IBC graph considering workload variations and the platform as a platform graph. Further, an *implementation-aware* IBC graph is created considering the design parameters (the number of pipes $p$, allocated processing cores for parallel execution per pipe $n_c^{//}$ and camera frame arrival period $f_h$).

#### IBC graph and implementation-aware graph

The IBC and implementation-aware graphs are modelled using an SADF model. Graph transformations to obtain an implementation-aware graph from the IBC graph are different for the different implementation choices and, as such, are explained in later sections. We choose SADF [129] as the formal MOC for our application as it enables us to: i) model dynamic behaviour and dependencies, analyse timing, and optimally map application (sub)tasks to the platform for maximising the effective utilisation of allocated resources; ii) relate latency and throughput of the dataflow graph to the control timing parameters $\tau$ and $h$, and thus combine dataflow analysis and mapping with control design parameters and QOC; iii) analyse inter-frame dependencies (captured as inter-frame dependence time $f_d$) through graph transformations (as explained in Section 4.4); and iv) to efficiently implement a runtime mechanism that manages necessary dynamic reconfiguration.

An SADF model is a tuple $(\Sigma, \mathcal{F})$ of scenarios $\Sigma$ and scenario sequences $\mathcal{F}$, as already explained in Section 2.3.1. The SADF model for our example IBC system is visualised in Fig. 4.4 (a). It is a variant of the model already used in Chapter 2, with an explicit image-signal preprocessing step and shorter actor names. The sensing and processing task receives the RAW camera image frames, which are processed in a sequence of steps to extract the state information required for the controller. The image-signal (pre-)processing (I) subtask converts the RAW

image in the Bayer domain to pixels in the RGB domain. (Sub-)tasks translate to actors in the dataflow graph, shown as circles in the figure. Data dependencies between (sub-)tasks translate to channels, shown as arrows. After the image processing, we detect the ROIs in the RGB image frames (D). ROI are processed (P), and, subsequently, the controller state (the lateral deviation $y_L$ in our LKAS case study) is computed by the ROI merging (M) subtask. The control algorithm (C) then computes the controller input $u[k]$ (steering angle $\delta_f$ in our LKAS case study) and feeds it to the actuation (A) task. The total number of ROI detected by D determines the workload $w_i$, i.e., $w_i = z$ in Fig. 4.4 (a). The workloads translate to variable token production and consumption rates in the graphs.

Graph transformations are required to analyse the parallel and/or pipelined implementations. This is, among others, because the typical mapping analysis tools assume that one actor can be bound to only one processing core. Fig. 4.4 (b) shows an implementation-aware graph for a non-pipelined parallelised implementation on two processors. It has two actor instances of the P subtask. The workload $w_i = z_1 + z_2$ in this case.

Each workload $w_i$ in an SADF is associated with an SDFG $\mathcal{G}_i$. An SDFG instance of Fig. 4.4 (b) is obtained by assigning values to parameters $e_j$ (the actor execution times) and $z_k$. E.g., assigning $z_1 = 3$, $z_2 = 3$, $e_i = 10$, $e_d = 5$, $e_p = 10$, $e_m = 3 \times (z_1 + z_2) = 18$, $e_c = 1$, $e_a = 1$ gives the SDFG for a workload of 6 ROI for mapping to two processors. There is one (labelled) initial token $t_1$ in the channel from actor

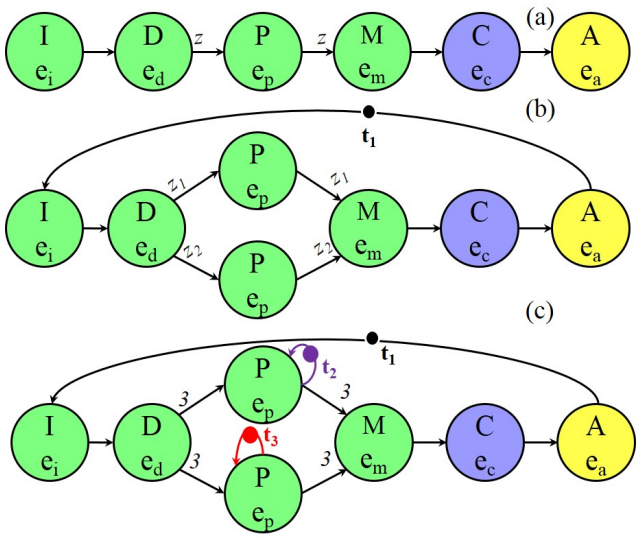



Figure 4.4: IBC SDFG: (a) graph structure. The rates $z$ indicate the workload $W$. (b) Implementation-aware graph for non-pipelined implementation on two cores (given platform allocation). (c) A (simplified) binding-aware graph for non-pipelined implementation on two cores for a workload of 6 ROI.

A to I. This channel, with its single initial token, enforces a non-pipelined execution of the control loop. All actors in Fig. 4.4 (a) have repetition-vector entries of 1, except actor $P$, which as a repetition-vector entry $z$; and all actors in Fig. 4.4 (b) have repetition-vector entries of 1, except the two $P$ instances that have entries $z_1$ and $z_2$ respectively.

Fig. 4.4 does not show the language of allowed scenario sequences. In the LKAS case, all possible workload sequences are allowed.

#### Platform graph

A platform, e.g. the CompSOC MPSoC shown in Fig. 1.5, is modelled as a platform graph that captures processing resources, and other relevant aspects such as memories and connections, with their processing and access latencies, data rates, etc. The details needed for the model depend on the used mapping flow. For the sake of explaining SPADE, we assume the platform is simply abstracted as a set of tiles. A tile $\mathcal{T}_i$ abstracts a resource with the processor type $pt_i$ that determines the execution time of actors bound to the tile. The CompSOC instance shown in Fig. 1.5 has three tiles. Two of these tiles have a microblaze processor type. The third tile is a memory tile that does not play a role in further explanations. Also, the connections are abstracted for the sake of simplicity. Hence, the platform is abstracted as a 2-node platform graph without any connections. Note that the used SDF3 mapping flow does support the modelling of memories and connections, including their timing, and takes these into account in the mapping optimisation.

A platform allocation determines the resources that are allocated to a task or to an application. Resources that are allocated may include the number of tiles or processors, or parts of processors (e.g. slots in a time-division multiplexing (TDM) frame in CompSOC), and types of processors, e.g. graphical processing unit (GPU), ARM, and microblaze. For our running LKAS example, an allocation consists only of the number of tiles of a specific processor type.

### 4.3.2 Analysis and design

We map the implementation-aware IBC graph for each workload $w_i \in W$ to the platform graph to obtain the *binding-aware graph* $\mathcal{G}_i^b$ (further explained below) using the SDF3 mapping flow [122]. A throughput and latency analysis of $\mathcal{G}_i^b$ yields the control timing parameters $\tau_i$ and $h_i$ for the workload scenario $s_i$. Controllers are designed for each workload scenario $s_i$ using the computed timing parameters $(\tau_i,\ h_i)$ to obtain the controller feedback and feedforward gains $(K_i,\ F_i)$. System-scenario identification is then performed to identify the set of system scenarios for runtime implementation. For a pipelined implementation, inter-frame dependence time $f_d$ (as explained in Section 4.5.4) is also computed using the throughput analysis.

**System mapping and mapping configurations**

System mapping refers to the mapping of the IBC application (modelled as an SADF) to the given platform (modelled as a platform graph). Note that for each workload scenario $s_i$, we can have multiple mapping options for the given platform allocation. The throughput and latency of each of these mapping options would be different. The concrete problem is to find the mapping of $s_i$ to the given platform allocation that maximises throughput. Any design flow that does the (Pareto-)optimal mapping of an application to a platform while maximising throughput can be used.

We use the SDF3 mapping flow [123] as it optimises the resource usage, memory load and communication load for mappings (to the extent that these aspects are considered in the models), and embeds state-of-the-art throughput analysis techniques. Mapping an $s_i$ (modelled as an SDFG $\mathcal{G}_i$) to a platform graph generates a binding-aware SDFG $\mathcal{G}_i^b$. $\mathcal{G}_i^b$ is an SDFG that models the mapping of the implementation-aware graph to the platform graph, where each actor in the SDFG is bound to a tile in the platform graph. For the ordering of execution of actors bound to the same tile, a static-order schedule is encoded in $\mathcal{G}_i^b$.

Fig. 4.4 (c) shows a simplified binding-aware graph for the 6-ROI workload scenario of the running example, bound to two tiles. It encodes two static-order schedules: IDP$^3$MCA for one iteration of the graph on one core and P$^3$ for one graph iteration on the second core. Self-loops with a single token need to be added to the two parallelised P actors to model the binding of the actor to a particular core and to enforce sequential execution of the P firings on each of the two cores. This suffices to encode the schedules. The graph is simplified in the sense that SDF3 encodes many more aspects in the binding-aware graph, such as memory accesses and interprocessor communication.

A *mapping configuration* $\chi_{s_i}^m$ refers to the binding of $s_i$ to the platform and its execution schedule represented in a binding-aware SDFG. The SPADE flow tries to minimise the number of cores used even if a given number of cores is allocated. This happens naturally when we map our SDFGs to the platform using the SDF3 tool, as SDF3 gives a Pareto-optimal mapping that minimises utilisation.

**Timing analysis - computing $f_d, \tau_i$ and $h_i$**

The computation of inter-frame dependence time $f_d$ is specific for pipelined implementation and is explained later in Section 4.5.4. In this subsection, we explain how we compute the throughput and latency of the SADF and relate it to the control timing parameters $\tau_i$ and $h_i$ for a workload scenario $s_i$. Note that the state-of-the-art SADF analysis uses (max, +) algebra [9] and the definitions needed for the computation of throughput have already been explained in Chapter 2. In this subsection, we summarise the relevant definitions for our analysis. For detailed explanations, the reader is referred to [4].

A time-stamp vector $\boldsymbol{\gamma_0}$ captures the availability times the initial tokens. The

production times of the final tokens resulting from the execution of a scenario $s$ are then $\boldsymbol{\gamma_1} = \boldsymbol{G_s}\boldsymbol{\gamma_0}$, where $\boldsymbol{G_s}$ is the scenario (or state) matrix of $s$. For the binding-aware scenario SDFG corresponding to 6 ROI, introduced in Fig. 4.4 (c), $\boldsymbol{\gamma_0} = [000]^T$. That is, the three initial tokens are all available at time 0. Scenario matrix $\boldsymbol{G_s}$ captures the dependencies and corresponding delays between the initial and final tokens. For the running example, $\boldsymbol{G_s}$ equals

$$\begin{bmatrix} e_i+e_d+3e_p+e_m+e_c+e_a & e_m+e_c+e_a & e_m+e_c+e_a \\ e_i+e_d+3e_p & 3e_p & -\infty \\ e_i+e_d+3e_p & -\infty & 3e_p \end{bmatrix}$$

Entry $ij$ in this matrix contains the time delay from consuming token $t_j$ to reproducing token $t_i$ in one iteration of the graph. The top left entry thus indicates the delay to reproduce the final token $t_1$ on the A-I channel. The two $3e_p$ entries show that the three firings of the two P actors are sequentialized. The two $-\infty$ entries indicate that the two self-loop tokens of the two P actors, $t_2$ and $t_3$, are independent. The other entries capture the delay from $t_1$ to the self-loop tokens $t_2$ and $t_3$ and the delay from $t_1$ to $t_2$ and $t_3$.

With the concrete actor execution times given earlier, this results in the following concrete matrix:

$$\begin{bmatrix} 65 & 50 & 50 \\ 45 & 30 & -\infty \\ 45 & -\infty & 30 \end{bmatrix}$$

The production times after execution of scenario $s$ are then obtained from $\boldsymbol{\gamma_1} = \boldsymbol{G_s}[000]^T = [\max(65,50,50)\max(45,30,-\infty)\max(45,-\infty,30)]^T = [654545]^T$. Note that the matrix multiplication in this analysis is the (max, +) matrix multiplication. The analysis shows that the three tokens in the binding-aware graph of Fig. 4.4 (c) are reproduced after 65, 45, and 45 time units, respectively.

The computation of token time-stamp vectors $\boldsymbol{\gamma_k}$, output time-stamp vectors $\boldsymbol{p}_k$, and throughput $\nu$ is explained in Section 2.3.1 in Eq. (2.8), Eq. (2.9), and Eq. (2.10). For the LKAS scenarios, the output is produced by the actor A, meaning that the output production time is equal to the production time of the token on the channel from A to I. This means that the $\boldsymbol{H_s}$ matrix for computing the output time stamps is equal to [655050], corresponding to the first row of $\boldsymbol{G_s}$. The production time of the first output $\boldsymbol{p}_0 = [655050][000]^T = [65]$. The throughput $\nu$ of this particular scenario is 1/65.

Latency is the maximum (worst-case) time taken to complete one iteration. Given an initial state $\boldsymbol{\gamma_0}$, the latency of a scenario sequence $\bar{s}$ relative to a period $\mu$ is defined as

$$\mathscr{L}(\bar{s},\boldsymbol{\gamma_0},\mu) = \max_{k\geq 0} \boldsymbol{p}_k - \mu k \ .$$

For the infinite execution of the 6-ROI scenario SDFG of Fig. 4.4 (c), the latency, relative to the period equalling the inverse of the throughput, is 65. We omit the details of the computation, referring the reader to [4]. But the results should not be surprising given the timing analysis of the scenario execution given earlier. Note that the inverse of throughput and latency are equal in this case due to the model with one initial token on the A-I channel that enforces the non-pipelined execution of the SDFG. We exploit such modelling tricks in our model transformations for mapping and pipelined implementation (as explained in later sections).

The sensor-to-actuator delay $\tau_i$ and the sampling period $h_i$ for the workload scenario $s_i$ that we need for controller design are computed from the binding-aware graph $\mathcal{G}_i^b$ that is obtained from mapping the implementation-aware graph onto the allocated resources (as explained earlier and elaborated in Section 4.5). The two values are computed as follows.



$$\tau_i = \mathcal{L}(s_i^\omega, \mathbf{0}, 1/\nu(\mathcal{G}_i^b)),\ h_i = \lceil \frac{\tau_i}{f_h \times p} \rceil f_h, \tag{4.2}$$

where $\mathbf{0}$ is the zero vector, $f_h$ is the camera frame arrival period, and $p$ is the number of pipes in the pipelined parallelism implementation. The delay $\tau_i$ of scenario $s_i$ is the latency of executing that scenario repetitively after mapping it onto the platform, with respect to the throughput obtained from that mapping and assuming that initial tokens are available at time 0. For the computation of the effective frame processing period $h_i$, $\lceil \frac{\tau_i}{f_h} \rceil$ computes the number of frame periods within the time-interval $\tau_i$. By dividing by the number of pipes $p$, rounding up, and multiplying with the frame period $f_h$, one obtains the effective sampling period for the particular scenario implementation. For the infinite execution of the 6-ROI scenario SDFG of Fig. 4.4 (c), assume $f_h = \frac{1}{60}$ s and $p = 1$. Then, $\tau_i = 65$ ms, in line with the earlier latency analysis, and $h_i = 66.7$ ms. Further details on how the SPADE flow uses $\tau_i$ and $h_i$ are provided in Section 4.5.

**Controller design and control configurations**

The LKAS case study we consider is a single-input single-output (SISO) system. We discretize the IBC system model in Eq. 4.1 using $(\tau_i,\ h_i)$, computed for the binding-aware SDFG $\mathcal{G}_i^b$ for the workload scenario $s_i$. Let $p$ be the number of pipes used in the implementation (as reflected in the binding-aware graph), where non-pipelined implementation corresponds to $p = 1$. We assume that $u[-1] = 0$ and define new system states $z[k] = \begin{bmatrix} x[k] & u[k-(p-1)] & \cdots & u[k-1] \end{bmatrix}^T$ with $z[0] = \begin{bmatrix} x[0] & 0 & \cdots & 0 \end{bmatrix}^T$ to obtain a higher-order augmented system as follows:

$$\begin{aligned} z[k+1] &= A_{aug,s_i} z[k] + B_{aug,s_i} u[k], \\ y[k] &= C_{aug} z[k] + D_c u[k], \end{aligned} \tag{4.3}$$

where $A_{aug,s_i}$, $B_{aug,s_i}$, and $C_{aug}$ are augmented system matrices. The computation of $A_{aug,s_i}$, $B_{aug,s_i}$, and $C_{aug}$ varies for the non-pipelined and pipelined implementation choices and as such is explained in the later sections. A check for controllability [37] is done for the augmented system. If the system is not controllable, controllability decomposition is done to obtain a controllable subsystem.

We can then apply standard control-design techniques [37] for the augmented system models in Eq. 4.3. We use a *state-feedback* controller $u[k]$ of the following form:

$$u[k] = K_i z[k] + F_i r_{ref} \tag{4.4}$$

where $K_i$ is the state-feedback gain and $F_i$ is the feedforward gain both designed for the workload scenario $s_i$. $r_{ref}$ is the constant reference value for the controller.



We design the gains using the optimal linear quadratic regulator (LQR) [37]. Note that any other state-of-the-art control-design technique can also be used for designing these gains. For each workload scenario $s_i$, we then define a *control configuration* $\chi^c_{s_i}$ as a tuple $\chi^c_{s_i} = (h_i, \tau_i, K_i, F_i)$.

#### System-scenario identification, system configurations and stability

System-scenario identification is done to limit the number of switching scenarios during runtime implementation. It is possible for multiple workload scenarios to have the same sensor-to-actuator delay and/or sampling period due to implementation constraints like platform allocation and camera frame rate [86].

For the non-pipelined implementation, a system scenario $s_s$ abstracts multiple workload scenarios $s_i$ such that for $h_s = n \times f_h$, for frame arrival period $f_h$ and some $n > 0$, $(h_s - f_h) < h_i \leq h_s$. That is, we aggregate workload scenarios based on $h_s$. Then, for the aggregated workload scenarios $s_i$, we choose $\tau_s$ to be the maximum among the $\tau_i$. $\mathcal{G}^b_s$, $K_s$ and $F_s$ are then re-designed for the $(\tau_s,\ h_s)$ identified for the system scenarios $s_s$. We design $\mathcal{G}^b_s$ by assigning $\tau = \tau_s$ and $h = h_s$ to the corresponding implementation-aware graph and verifying the existence of a mapping that satisfies $\tau_s$ and $h_s$. A control configuration $\chi^c_{s_s} = (h_s, \tau_s, K_s, F_s)$ is then derived following the approach outlined earlier for workload scenarios. Only system scenarios are then considered for defining the system configurations $\chi^s_{s_s}$, which is a combination of control configuration $\chi^c_{s_s}$ and mapping configuration $\chi^m_{s_s}$, i.e., $\chi^s_{s_s} = (\mathcal{G}^b_s,\ h_s,\ \tau_s,\ K_s,\ F_s)$. The system-scenario identification for pipelined implementation is explained in Section 4.5.4.

At runtime, the system scenarios switch based on the image-workload variations and/or platform load. This switching behaviour can lead to system instability. Therefore, we must *guarantee* stability of the overall system while improving QoC (already explained in Section 2.2.4).

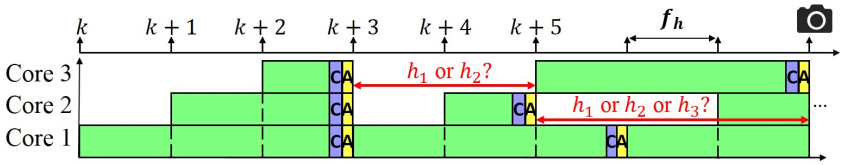



Figure 4.5: Challenges in pipelined implementation due to switching when $h_i$ is a multiple of $f_h$.

### 4.3.3 Runtime implementation

At design time, the system configurations $\chi^s_{s_s}$ are stored in a LUT in platform memory. During runtime, for every arriving input image frame, we compute the workload $w_i$ (e.g. through the ROI detection task D) and choose the correct system scenario $s_s$ associated with this workload from the LUT. System configuration $\chi^s_{s_s}$ of the corresponding system scenario $s_s$ is loaded from the LUT. Dynamic runtime reconfiguration is typically needed since there can be a switching behaviour between system configurations due to image-workload variations. For the non-pipelined implementation and the pipelined implementation without resource sharing between pipes, dynamic runtime reconfiguration means that, if needed, a scheduler reconfigures the mapping $\mathcal{G}^b_s$, the time-triggering of the actuation task (that determines $\tau_s$) and the controller gain parameters ($K_s$ and $F_s$) based on the system scenario $s_s$ associated with the image workload from the LUT. The overhead cost for this reconfiguration needs to be considered in the analysis model, e.g., for the LKAS example, as an additional execution time cost in the actor D (see Fig. 4.4).

Arbitrary switching and reconfiguration in the pipelined implementation are challenging. Let $p$ be the number of pipes, and $h_s$ be the periods per system scenario. If we restrict $h_s$ to be a multiple of $f_h$, the number of periods possible due to arbitrary switching considering image-workload variations only grows linearly with $f_h$ and $p$. However, if we do not restrict $h_s$ and allow it to take arbitrary values, the number of periods possible grows exponentially. E.g. assume that we have three periods $h_1$ (= $f_h$ for workload $w_1$), $h_2$ (= $2f_h$ for workload $w_2$), and $h_3$ (= $3f_h$ for workload $w_3$) due to image-workload variations. For simplicity, let us assume that $\tau_s = h_s$ and a given three-core platform allocation with three pipes. Consider the case shown in Fig. 4.5 where the image frames $k$, $k+1,\ldots,$ $k+i$ have workload $w_3$, $w_2$, $w_1$, $w_3$, $w_1$, $w_3$, and so on. When multiple control computations complete at the same time, e.g., just before frame $k+3$ is captured, the actuation should be coordinated among the cores. Further, the controller for the image frame $k+4$ should ideally be designed using the discretized model considering $h_1 = f_h$ if the period is defined as the time between two consecutive starts of the sensing task. However, it should also take into account that there was no actuation just before this, i.e., the previous actuation was at the time $t-2f_h$. So, if the period was defined as the time between two consecutive actuations, then the period for the con-

troller for the frame at $k+4$ is $2f_h$. A similar situation exists for frame $k+5$. Such behaviours add to the complexity of the design space to be explored. The main challenge, however, is proving the stability of the ensuing switched system with these behaviours. Also, modelling these behaviours for the control design is far from trivial.

For the scope of this work, we enforce a constant sampling period $h_{eff}$ for the overall pipelined implementation. A constant sampling period helps to limit the design space to be explored and handle the dynamic reconfiguration with less run-time overhead. As explained, the sensor-to-actuator delay $\tau_s$ is constant per identified system scenario $s_s$. Consequently, two system scenarios $s_1$ and $s_2$ have the control timing parameters $(h_{eff}, \tau_1)$ and $(h_{eff}, \tau_2)$.

A similar challenge exists for a pipelined implementation with resource sharing between pipes, where reconfiguring the mapping dynamically is non-trivial. Resource sharing between pipes increases the design space to be explored for considering the possible reconfiguration options. For the scope of this work, dynamic reconfiguration for pipelined implementation with resource sharing between pipes comprises a static mapping where actors are switched on and off considering image workload variations, and choosing the controller gains dynamically from the LUT by the control-computation task based on the system scenario considering the latest state measurement available.

The SPADE flow is not restricted to the mentioned design and implementation choices. Its key feature is that pipelining and parallellism are integrally considered. As we will see, this provides benefits in the achievable QoC. Other controller design and implementation choices can be integrated as long as appropriate timing analysis and stability guarantees can be provided.

## 4.4 Model transformations

This section explains the model transformations required for modelling, analysing, and mapping the IBC system using SADF. The model transformations are required to obtain the implementation-aware IBC graph from the IBC graph for the given design parameters, as illustrated in Fig. 4.3. Our model transformations consist of maximising parallelism, creating a pipe, replicating pipes to implement pipelining, introducing camera-awareness, introducing workload-awareness, modelling inter-frame dependencies, and re-timing of actor execution times. For each workload scenario, we assume that the sensing and processing task is modelled as an SDFG $\mathcal{G}_S$, the control computation task is modelled as an SDFG $\mathcal{G}_C$, and the actuation task is modelled as an SDFG $\mathcal{G}_A$. The graphs $\mathcal{G}_S$, $\mathcal{G}_C$, and $\mathcal{G}_A$ should have identifiable source and sink actors $a_{src,i}$ and $a_{snk,i}$, $i \in \{S, C, A\}$. A source is an actor without any incoming edges and a sink is an actor without any outgoing edges. We moreover enforce that $\rho(a_{src,i}) = \rho(a_{snk,i}) = 1$. Having identifiable source and sink actors with repetition-vector entries equal one ensures well-formedness for our model transformations. Note that an SDFG with a single actor

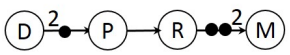
satisfies the assumptions. The source and sink actors should be identical across all workload SDFGs in an application IBC SADF.

To maximize opportunities to speed up the computations in the control loop, we want to maximize parallelism in graphs $\mathcal{G}_S$, $\mathcal{G}_C$, and $\mathcal{G}_A$. Automatically extracting task- and data parallelism in computations is challenging. So, in general, it is up to the designer to maximize parallelism in the three mentioned graphs. But given an SDFG of a workload scenario, it is possible to maximize data parallelism by transforming the SDFG to a homogeneous synchronous dataflow graph (HSDFG) [73, 121]. Essentially, this transformation replicates actors with a repetition-vector entry greater than one into multiple actors (as many as the repetition-vector entry of the actor for the SDFG) with each a repetition-vector entry one in the HSDFG. A platform-aware mapping such as implemented in the SDF3 tool [123] then clusters actors of the HSDFG per processor in the given platform allocation for maximising throughput. A disadvantage of this approach, however, is the scalability of the mapping and performance analysis that depends on the number of actors.

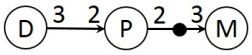


Another option is to replicate the parallelisable actors as many times as meaningful given the platform allocation. That is, we transform an SDFG $\mathcal{G}$ via a transformation $RepA(\mathcal{G}, \varphi)$ that preserves the number and timing of firings in a single iteration of the original graph $\mathcal{G}$ in the transformed graph, where $\varphi$ is the replication vector with size equal to the number of actors in $\mathcal{G}$ and where each element $\varphi(a)$ represents the number of times an actor $a$ needs to be replicated. A straightforward replication vector can then be defined using the repetition vector $\rho$ and the maximum number of processing cores allocated for parallel execution of tasks per pipe $n_c^{//}$, as $\varphi(a) = min(\rho(a), n_c^{//}),\ a \in \mathcal{A}$. Often, this transformation is relatively straightforward, but a definition that works in general is not obvious. The challenge when replicating actors is to accurately model the transformations of channels, production and consumption rates, and initial tokens such that the functional and timing behaviour of the original graph is preserved. Fig. 4.6 gives some example transformations, including Gantt charts that illustrate that actor firings and their timing are preserved. We leave a generic definition (and the proof that such a transformation exists in general and preserves functionality and timing) as future work. Note that the SDFG-to-HSDFG transformation of [73, 121] is an instance of $RepA$ when the replication vector is chosen equal to the repetition vector.

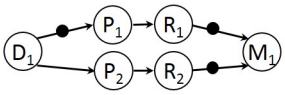
For the remainder, assume that $\mathcal{G}_S$, $\mathcal{G}_C$, and $\mathcal{G}_A$ are the graphs obtained after maximizing parallelism. The **Create pipe** $Pipe(\mathcal{G}_S, \mathcal{G}_C, \mathcal{G}_A)$ transformation creates a model for a single pipe by adding a delay actor and channels between the sinks and sources of $\mathcal{G}_S$ and $\mathcal{G}_C$, $\mathcal{G}_C$ and $\mathcal{G}_A$, $\mathcal{G}_A$ and delay, and delay and $\mathcal{G}_S$. The execution time of the delay actor is set to zero, and one initial token is added to the channel between the delay actor and the source of $\mathcal{G}_S$ to enforce sequential implementation of the pipe (see Fig. 4.7). The latency of the resulting SDFG can be configured by an appropriate choice of the execution time of the delay actor (which can be set using the re-timing transformation given in Def. 6). The model transfor-



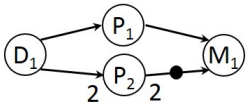

(a) $\mathcal{G}_1$



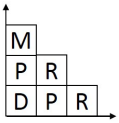

(b) $\mathcal{G}_2$



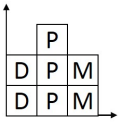

(c) $RepA(\mathcal{G}_1, [1\ 2\ 2\ 1])$




(d) $RepA(\mathcal{G}_2, [1\ 2\ 1])$




(e) Gantt chart for 4.6(a) and 4.6(c)




(f) Gantt chart for 4.6(b) and 4.6(d)

Figure 4.6: Examples of the replicate actors $RepA$ transformation. The Gantt charts cover one iteration of the corresponding graph and assume actor execution times to be 1. Subscripts resulting from $RepA$ transformations are omitted for brevity.

mation results in an SDFG whose latency is equal to the inverse of the throughput.

**Definition 1.** ***(Create pipe*** $Pipe(\mathcal{G}_S, \mathcal{G}_C, \mathcal{G}_A)$***)*** *Transformation* $Pipe(\mathcal{G}_S, \mathcal{G}_C, \mathcal{G}_A)$ *creates a single pipe and sequentialises the graph execution (by restricting pipelining).* $Pipe(\mathcal{G}_S, \mathcal{G}_C, \mathcal{G}_A) = (\mathcal{A}', \mathcal{C}', e', r'_p, r'_c, i')$ *with*

$$
\begin{aligned}
\mathcal{A}' &= \mathcal{A}_S \cup \mathcal{A}_C \cup \mathcal{A}_A \cup \{delay\},\\
\mathcal{C}' &= \mathcal{C}_S \cup \mathcal{C}_C \cup \mathcal{C}_A \cup \{c_1 = (a_{snk,S}, a_{src,C}),\ c_2 = (a_{snk,C}, a_{src,A}),\\
&\qquad\qquad c_3 = (a_{snk,A}, delay),\ c_4 = (delay, a_{src,S})\},\\
e' &= e_S \cup e_C \cup e_A \cup \{(delay, 0))\},\\
r'_p &= r_{p_S} \cup r_{p_C} \cup r_{p_A} \cup \{(c_1,1),(c_2,1),(c_3,1),(c_4,1)\},\\
r'_c &= r_{c_S} \cup r_{c_C} \cup r_{c_A} \cup \{(c_1,1),(c_2,1),(c_3,1),(c_4,1)\},\\
i' &= i_S \cup i_C \cup i_A \cup \{(c_1,0),(c_2,0),(c_3,0),(c_4,1)\}.
\end{aligned}
$$

The $Pipe$ transformation is essential to compute sensor-to-actuator delay $\tau$ for our implementations. To compute $\tau_i$ for a workload scenario $s_i$: i) compute $Pipe(RepA(\mathcal{G}_{S_i}, \varphi_{S_i}), RepA(\mathcal{G}_{C_i}, \varphi_{C_i}), RepA(\mathcal{G}_{A_i}, \varphi_{A_i}))$; ii) map the transformed graph to the given platform allocation to obtain the binding-aware graph $\mathcal{G}_i^b$; and iii) compute the latency of $\mathcal{G}_i^b$. This latency value is equal to $\tau_i$.

The replicate-pipe transformation is an intermediate step in the model transformation, where we replicate the entire pipe to enable pipelining (see Fig. 4.7, $RepP(g_1, 2)$). Since each actor can be mapped to only one processing core, implementing pipelining on multiple processing cores is challenging without replication

of a single pipe. $RepP(\mathcal{G},d)$ facilitate the modelling for a pipelined implementation with $d$ pipes. Recall that a pipe is created from $\mathcal{G}_S$, $\mathcal{G}_C$ and $\mathcal{G}_A$ (see Definition 1). These subgraphs in a single pipe are all replicated $d$ times by the replicate-pipe transformation. The $RepP$ transformation does not use the specifics of the graph it is applied to. It generically replicates the entire SDFG the specified number of times.

**Definition 2.** ***(Replicate pipe** $RepP(\mathcal{G},d)$**)*** *Let* $\mathcal{G} = (\mathcal{A}, \mathcal{C}, e, r_p, r_c, i)$ *be an* SDFG. *Transformation* $RepP(\mathcal{G},d)$ *replicates the* SDFG $d$ *times resulting in* $RepP(\mathcal{G},d) = (\mathcal{A}', \mathcal{C}', e', r'_p, r'_c, i')$ *with*

$$\mathcal{A}' = \bigcup_{a\in\mathcal{A}} \{a_j \mid 1 \le j \le d\}^1,$$
$$e' = \{(a_j, e(a)) \mid a \in \mathcal{A},\ 1 \le j \le d\},$$
$$\mathcal{C}' = \bigcup_{c\in\mathcal{C}} \{c_j \mid 1 \le j \le d\},$$
$$i' = \{(c_j, i(c)) \mid c \in \mathcal{C},\ 1 \le j \le d\},$$
$$r'_p = \{(c_j, r_p(c)) \mid c \in \mathcal{C},\ 1 \le j \le d\},$$
$$r'_c = \{(c_j, r_c(c)) \mid c \in \mathcal{C},\ 1 \le j \le d\}.$$

The three following transformations - adding camera-awareness, workload-awareness and inter-frame dependencies - assume that the replicate-pipe transformation has been performed with replication factor $d$ on a single pipe created with transformation $Pipe$, optionally preceded by replicate-actor transformations (which do not affect these transformations).

The **Camera-awareness** transformation is an intermediate step in our model transformations. It adds a camera actor with execution time equal to the inverse of the given camera frame rate (the frame arrival period $f_h$), a self-edge with an initial token to model the frame arrival, and channels from the camera actor to the $d$ replicated source actors $a_{src,S_j}$ with the channel consumption rate equal to the number of replications $d$ (see Fig. 4.7, $Cam(g_2,2)$). The number of initial tokens in these channels are set to enforce an ordering of the pipes in the pipelined implementation.

**Definition 3.** ***(Camera-awareness** $Cam(\mathcal{G},d)$**)*** *Let* $\mathcal{G} = (\mathcal{A}, \mathcal{C}, e, r_p, r_c, i)$ *be the* SDFG $RepP(Pipe(\mathcal{G}_S,\mathcal{G}_C,\mathcal{G}_A), d)$. *Transformation* $Cam(\mathcal{G},d) =$

[1] This union replicates every actor in the original graph $d$ times. For every actor $a$ in $\mathcal{A}$, we create $a_1,\ldots, a_d$ actors in the new graph.

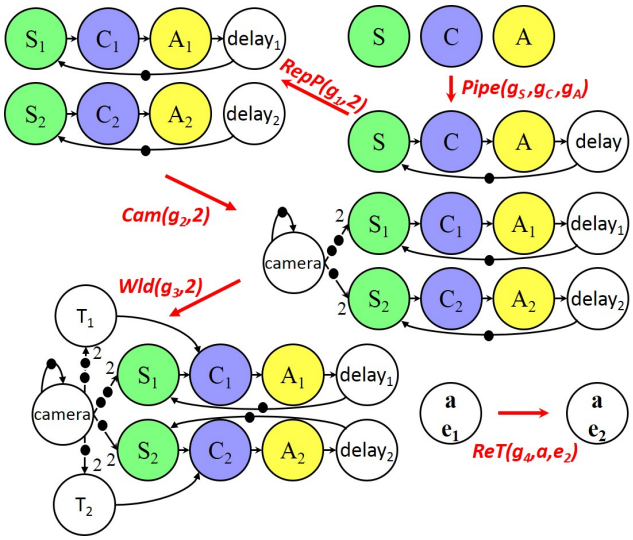



Figure 4.7: Illustration of model transformations.

$(\mathscr{A}', \mathscr{C}', e', r'_p, r'_c, i')$ *enforces a camera frame rate, with*

$$\mathscr{A}' = \mathscr{A} \cup \{camera\},\ e' = e \cup \{(camera, f_h)\},$$
$$\mathscr{C}' = \mathscr{C} \cup \{(camera)^2\} \cup$$
$$\{c_j = (camera, a_{src,S_j}) \mid 1 \le j \le d\},$$
$$r'_p = r_p \cup \{((camera)^2, 1)\} \cup \{(c_j, 1) \mid 1 \le j \le d\},$$
$$r'_c = r_c \cup \{((camera)^2, 1)\} \cup \{(c_j, d) \mid 1 \le j \le d\},$$
$$i' = i \cup \{((camera)^2, 1)\} \cup \{(c_j, d - j + 1) \mid 1 \le j \le d\}.$$

The **Workload-awareness** transformation is a step in the model transformations performed on graph $Cam(RepP(Pipe(\mathscr{G}_S, \mathscr{G}_C, \mathscr{G}_A), d), d)$. Due to image-workload variations, the sensing task's runtime execution times are varying. Also, for the SPADE implementation, the system scenarios may abstract multiple workload scenarios with varying execution times for the sensing task. However, the SPADE controller design requires a constant sensor-to-actuator delay per system scenario for the implementation. The $Wld$ transformation enforces a constant sensor-to-actuator delay for our implementation. The $Wld$ transformation adds actors $T_j$ with an incoming channel from the camera actor and an outgoing channel to the (replicated) source actor $a_{src,C_j}$ of the computation SDFG $\mathscr{G}_C$. The consumption rate and initial tokens for the channels from camera actor to $T_j$ are the same as for the channel from camera actor to source actors in the $Cam$ transformation, again to enforce ordering in the pipelined execution (see Fig. 4.7, $Wld(g_3, 2)$). The $T_j$ actors create a path in parallel to the $\mathscr{G}_S$ graph instances. By

setting the execution time of these added $T_j$ actors to an appropriately large value, a constant sensor-to-actuator delay can be enforced. The $Wld$ transformation sets the execution time to 0. The execution-time value can be updated when needed in the SPADE flow by the re-timing transformation introduced below.

**Definition 4.** ***(Workload-awareness** $Wld(\mathcal{G},d)$**)*** *Let* $\mathcal{G}=(\mathcal{A},\mathcal{C},e,r_p,r_c,i)$ *be the* SDFG $Cam(RepP(Pipe(\mathcal{G}_S,\mathcal{G}_C,\mathcal{G}_A),d),d)$. *Transformation* $Wld(\mathcal{G})=(\mathcal{A}',\mathcal{C}',e',r_p',r_c',i')$, *with*

$$
\begin{aligned}
\mathcal{A}' &= \mathcal{A} \cup \{T_j \mid 1 \le j \le d\},\\
e' &= e \cup \{(T_j,0) \mid 1 \le j \le d\},\\
\mathcal{C}' &= \mathcal{C} \cup \{cT_j=(camera,T_j) \mid 1 \le j \le d\} \cup\\
&\qquad \{TC_j=(T_j,a_{src,C_j}) \mid 1 \le j \le d\},\\
r_p' &= r_p \cup \{(cT_j,1) \mid 1 \le j \le d\} \cup\\
&\qquad \{(TC_j,1) \mid 1 \le j \le d\},\\
r_c' &= r_c \cup \{(cT_j,d) \mid 1 \le j \le d\} \cup\\
&\qquad \{(TC_j,1) \mid 1 \le j \le d\},\\
i' &= i \cup \{(cT_j,d-j+1) \mid 1 \le j \le d\} \cup\\
&\qquad \{(TC_j,0) \mid 1 \le j \le d\}.
\end{aligned}
$$

The **inter-frame-dependency** transformation adds channels to enforce the dependencies for actor firings between two consecutive pipes. E.g., an actor $b_j$ that executes in the $k$-th pipe might depend on the completion of execution of an actor $a_i$ that executes in the $(k-1)$-th pipe. This transformation is optionally done after $Cam$ (and has no further effect on the definition of the earlier transformations). An example for this transformation is illustrated in Fig. 4.8. Ideally, our model transformations ensure that the inverse throughput of the implementation-aware graph is equal to the execution time of the camera actor. E.g. if $e$(camera) $= f_h$, then the throughput of the implementation-aware graph is equal to (or limited by) the camera frame rate $\frac{1}{f_h}$. Now, the *ifd* transformation allows the throughput to be limited also by the inter-frame dependencies. The inverse throughput of the implementation-aware graph will then be equal to the maximum of $e$(camera) and the inter-frame dependence time $f_d$ (as explained later in Section 4.5.4).

**Definition 5.** ***(Inter-frame dependency** $ifd(\mathcal{G},a,b,d)$**)*** *Let* $\mathcal{G}=(\mathcal{A},\mathcal{C},e,r_p,r_c,i)$ *be the* SDFG $Cam(RepP(Pipe(\mathcal{G}_S,\mathcal{G}_C,\mathcal{G}_A),d),d)$. *Transformation* $ifd(\mathcal{G},a,b,d)=(\mathcal{A},\mathcal{C}',e,r_p',r_c',i')$ *adds inter-frame dependencies between actors* $a_i$ *and* $b_j$, *for* $a_i,b_j \in \mathcal{A}$, *with*

$$
\begin{aligned}
\mathcal{C}' &= \mathcal{C} \cup \{c_d=(a_d,b_1)\} \cup \{c_j=(a_j,b_{j+1}) \mid 1 \le j < d\},\\
r_p' &= r_p \cup \{(c_d,\rho(b_1))\} \cup \{(c_j,\rho(b_{j+1})) \mid 1 \le j < d\},\\
r_c' &= r_c \cup \{(c_d,\rho(a_d))\} \cup \{(c_j,\rho(a_j)) \mid 1 \le j < d\},\\
i' &= i \cup \{(c_d,\rho(a_d)))\} \cup \{(c_j,0) \mid 1 \le j < d\}.
\end{aligned}
$$

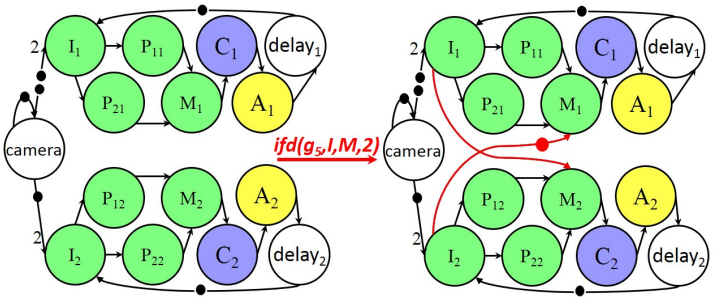



Figure 4.8: Illustration of inter-frame dependencies between the actors I and M. The channels added by $ifd(g_5, I, M, 2)$ are shown in red.



During SPADE analysis, the $\tau$ and $h$ values are updated based on our implementation choices. The **re-timing** transformation helps to update the execution time of the actors (camera, delay, and T) in our models when required (see Fig. 4.7, $ReT(g_4, a, e_2)$).

**Definition 6.** ***(Re-timing** $ReT(\mathcal{G}, a, t)$**)*** *Let $\mathcal{G} = (\mathcal{A}, \mathcal{C}, e, r_p, r_c, i)$ be an SDFG. Transformation $ReT(\mathcal{G}, a, t) = (\mathcal{A}, \mathcal{C}, e', r_p, r_c, i)$ updates the execution time of actor $a \in \mathcal{A}$ to $t \in \mathbb{R}_{\geq 0}$, with*

$$e' = e \setminus \{(a, e(a))\} \cup \{(a, t)\}.$$

## 4.5 The SPADE flow revisited

This section makes the SPADE design flow (illustrated in Fig. 4.3 and introduced in Section 4.3) precise in the form of Algorithm 2, using the model transformations of Section 4.4. The model transformations are primarily used to construct implementation-aware graphs for workload and system scenarios. The integrated transformation from an IBC graph to an implementation-aware graph is captured in Algorithm 1. Due to platform resource constraints (the total number of available cores $n_c^{avl}$), design choices (number of pipes $p$, number of cores allocated per pipe $n_c^{//}$), application characteristics, inter-frame dependencies (inter-frame dependence time $f_d$) and the possible camera frame-arrival period ($f_h$), the effective implementation can be: i) a non-pipelined implementation without parallelised sensing; ii) a non-pipelined implementation with parallelised sensing; iii) a pipelined implementation without parallelised sensing; and iv) a pipelined parallelism implementation. Section 4.5.1 explains Algorithm 1 for constructing implementation-aware graphs. Section 4.5.2 elaborates SPADE in Algorithm 2. We then explain the refinements for non-pipelined and pipelined implementations in Sections 4.5.3 and 4.5.4. The differentiation between non-pipelined and pipelined implementation is mainly needed for control design and switching. Further, we

also explain the challenge due to inter-frame dependencies in a pipelined implementation in Section 4.5.4.

**Algorithm 1:** impAwGrTrans($\mathcal{G}, \tau, h, e_{CA}, p$)

**input :** $\mathcal{G}$, $\tau$, $h$, $e_{CA}$, $p$, *IFD* (the set of known inter-frame dependencies)
**output:** Implementation-aware graph $\mathcal{G}_I$

$\mathcal{G}_1 = ReT(\mathcal{G}, \text{delay}, (p \times h - \tau))$;
$\mathcal{G}_3 = Cam(RepP(\mathcal{G}_1, p), p)$;
$\mathcal{G}_{31} = ReT(\mathcal{G}_3, \text{camera}, h)$
**if** $p > 1$, *i.e. pipelining is allowed* **then**
  **foreach** $(a, b) \in IFD$ **do**
    $\mathcal{G}_{31} = ifd(\mathcal{G}_{31}, a, b, p)$;
  **end**
**end**
$\mathcal{G}_4 = Wld(\mathcal{G}_{31}, p)$;
$\mathcal{G}_I = ReT(\mathcal{G}_4, T_j, (\tau - e_{CA}))$, $1 \leq j \leq p$;

### 4.5.1 Implementation-aware graph transformation

In this section, we explain the steps needed to obtain the implementation-aware graph of a workload or system scenario and formalise those in Algorithm 1. The **input** SDFG is the SDFG of a single parallelised pipe of a single scenario, obtained after applying the $Pipe$ transformation (as explained in Section 4.4), as illustrated in Fig. 4.7. The other inputs to the algorithm are the delay $\tau$, period $h$, the total execution time of the control compute and actuation tasks $e_{CA}$, and the number of pipes $p$. Also, we assume that if there exist inter-frame dependencies, they are known as a subset of actors *IFD*, $IFD \subseteq \mathcal{A}^2$. An ordered pair of actors $(a, b) \in IFD$ models the inter-frame dependency between the actors $a$ and $b$. The **output** of the algorithm is the implementation-aware graph $\mathcal{G}_I$.

**Step 1** ensures that we can achieve a constant sensor-to-actuator delay during mapping by assigning the execution time of the delay actor as $p \times h - \tau$ in $\mathcal{G}$ to obtain $\mathcal{G}_1$. The delay actor fills up the time between when the actuation task (modelled by actor A) finishes its execution until the completion of one pipe (see Fig. 4.7). This ensures that each parallelised pipe, when mapped to the platform, can periodically execute with the period $p \times h$ and has a constant delay of $\tau$. If we have $p$ pipes, we can ensure that the effective control sampling period is $h$. **Step 2** replicates the single parallelised pipe model $\mathcal{G}_1$ $p$ times to model the pipelined execution and then adds camera-awareness to the graph $\mathcal{G}_1$ as explained in Section 4.4. **Step 3** updates the execution time of the camera actor in line with the sampling period $h$. Our model has to execute with the period $h$ even though the camera frame arrival period is $f_h$. $h$, however, is a multiple of $f_h$ so that we can align the arrival of camera frames with the sampling period.

If pipelining is allowed, i.e., if $p > 1$, and there exist inter-frame dependencies that are known as ordered pairs of actors *IFD*, the *ifd* transformation, explained in Section 4.4 and in Fig. 4.8, is applied for each of the dependencies (see **Step 4** and the for loop in **Step 5**). **Step 9** adds workload-awareness to the resulting graph $\mathcal{G}_3$, also explained in Section 4.4. The $Wld$ transformation adds the actors $T_j$ whose execution times should be equal to the sensor-to-actuator delay minus the execution times of the control compute and actuate tasks $e_{CA}$ (an input to the algorithm). We obtain the final refined implementation-aware graph $\mathcal{G}_I$ after updating the execution time of actors $T_j$ in **Step 10** with $e(T_j) = \tau - e_{CA}$ so that we can ensure that the control compute task starts at the right time and is not affected by the workload variations in sensing.



### 4.5.2 Unified SPADE flow for pipelined parallelism

The SPADE flow of Fig. 4.3 is made precise in Algorithm 2. Algorithm 2 captures the design-time formal modelling, analysis and design for the SPADE flow. The **outputs** of the design flow are the system configurations and the LUT for runtime implementation. Runtime implementation for SPADE has been explained earlier in Section 4.3.3. The **inputs** to the SPADE flow are the camera frame rate $f_h$, the number of pipes $p$, the number of cores for parallelism per pipe $n_c^{//}$, the application IBC SADF $(\Sigma, \mathcal{F})$ (satisfying the assumptions given in Section 4.4), and the total (maximum) number of available cores $n_c^{avl}$. 'Map $\mathcal{G}$' denotes the mapping of the SDFG graph $\mathcal{G}$ to the given platform allocation, as mentioned in **Step 2**. In our implementation, the mapping is done using the SDF3 [123] tool. But any mapping tool can be used that ensures a mapping onto the platform that guarantees the maximal throughput obtainable by the SADF being mapped.

We explain the steps in Algorithm 2 in relation to Fig. 4.3. The 'for loop' in **Step 3** derives, for each workload scenario $s_i$, the initial implementation-aware IBC graph $\mathcal{G}_{11_i}$ (Steps 4,5), mapping of the control compute and actuation tasks to obtain $\mathcal{G}^b_{CA_i}$ (Step 6), mapping of the initial implementation-aware graph $\mathcal{G}_{11_i}$ to obtain the binding-aware graph $\mathcal{G}^b_{11_i}$ (Step 7), and timing analysis for computing $\tau_i$ and $h_i$ (Steps 8, 9). If pipelining is enabled, we refine the implementation-aware graph using the timing analysis information and the implementation choice $p$ (Step 11) to compute the inter-frame dependence time for the scenario $f_d^i$ (Step 14). The 'for loop' in Step 3 is illustrated in Fig. 4.3 from the implementation-aware IBC graph node to the timing analysis block and back. The model transformations required to compute the implementation-aware graphs have already been illustrated in Fig. 4.7 and Fig. 4.8 and the transformation of Step 11 has been made precise in Algorithm 1.

**Steps 4 and 5** create a model of a single parallelised pipe $\mathcal{G}_{11_i}$, as explained in Section 4.4. The parallelisation transformations are optional. As explained in Section 4.4, the parallelisation may also be done manually. **Step 6** maps the control compute and actuate tasks to the given platform allocation to compute its ex-

**Algorithm 2:** SPADEFlow($f_h$, $p$, $n_c^{//}$, $(\Sigma, \mathscr{F})$, $n_c^{avl}$)

**input :** $f_h$, $p$, $n_c^{//}$, $(\Sigma, \mathscr{F})$ ( SADF), $n_c^{avl}$ (platform)
**output:** System configurations $\chi_{s_s}^{s}$, LUT

**Begin**
  Let 'Map $\mathscr{G}$' denote the mapping of an SDFG $\mathscr{G}$ to the given $n_c^{avl}$ cores using the SDF3 tool;
  **foreach** *workload scenario* $s_i \in \Sigma$ **do**
    $\varphi_{X_i} = min(\rho_{X_i}, n_c^{//})$, where $X_i \in \{S_i, C_i, A_i\}$ and $\rho_{X_i}$ is the repetition vector of $X_i$;
    $\mathscr{G}_{11_i} = Pipe(RepA(\mathscr{G}_{S_i}, \varphi_{S_i}), RepA(\mathscr{G}_{C_i}, \varphi_{C_i}), RepA(\mathscr{G}_{A_i}, \varphi_{A_i}))$;
    $\mathscr{G}^b_{CA_i} \leftarrow$ Map $Pipe(RepA(\mathscr{G}_{C_i}, \varphi_{C_i}), RepA(\mathscr{G}_{A_i}, \varphi_{A_i}))$;
    $\mathscr{G}^b_{11_i} \leftarrow$ Map $\mathscr{G}_{11_i}$;
    $\tau_i = \mathscr{L}(s_i^{\omega}, \mathbf{0}, \frac{1}{\nu(\mathscr{G}^b_{11_i})})$;
    $h_i = \lceil \frac{\tau_i}{f_h \times p} \rceil f_h$;
    **if** $p > 1$, *i.e. pipelining is allowed* **then**
      $\mathscr{G}_{I_i} = \text{impAwGrTrans}(\mathscr{G}_{11_i}, \tau_i, h_i, \frac{1}{\nu(\mathscr{G}^b_{CA_i})}, p)$;
      $\mathscr{G}_{32_i} = ReT(\mathscr{G}_{I_i}, \text{camera}, 0)$;
      $\mathscr{G}^b_{32_i} \leftarrow$ Map $\mathscr{G}_{32_i}$;
      $f_d^i = \frac{1}{\nu(\mathscr{G}^b_{32_i})}$;
    **end**
  **end**
  **if** $p > 1$, *i.e. pipelining is allowed* **then**
    $\tau_{wc} = \max_i \tau_i$; $h_{wc} = \max_i h_i$;
    $f_d = \max_i f_d^i$;
    $n_s = \max\left(\lceil \frac{f_d}{f_h} \rceil, 1\right)$;
    $n_{f_{wc}} = \lceil \frac{\tau_{wc}}{f_h} \rceil$;
    $p_{max} = \lceil \frac{n_{f_{wc}}}{n_s} \rceil$;
    $n_{c_{max}} = n_c^{//} \times p_{max}$;
    $$h_{min} = \begin{cases} n_s \times f_h, & \text{if } n_c^{avl} \geq n_{c_{max}}, \\ \left\lceil \frac{n_{c_{max}}}{n_c^{avl}} n_s \right\rceil \times f_h, & \text{otherwise;} \end{cases}$$
    $h_{eff} = \max(h_{min}, h_{wc})$;
  **end**
  Controller design and system-scenario identification:
    *if p>1*, see Sections 4.5.4 and 4.5.4;
    *else* see Sections 4.5.3 and 4.3.2;

28 $s_s \leftarrow$ identified system scenarios with $(\tau_s, h_s)$;
29 $\tau_{s_{wc}} = \max_s \tau_s$; $h_{s_{wc}} = \max_s h_s$;
30 $s_{s_{wc}} = \arg\max_{s_s} \tau_s$ ;
31 $p_s = \lceil \frac{\tau_{s_{wc}}}{h_{s_{wc}}} \rceil$ ;
32 **foreach** *identified system scenario* $s_s$ *with* $(\tau_s, h_s)$ **do**
33 $\mathcal{G}_{11_s} \leftarrow \mathcal{G}_{11_i}$ of the $s_i$ in $s_s$ with max $\tau_i$;
34 $\mathcal{G}^b_{CA_s} \leftarrow \mathcal{G}^b_{CA_i}$ of the $s_i$ in $s_s$ with max $\tau_i$;
35 $\mathcal{G}_{I_s}$ = impAwGrTrans$(\mathcal{G}_{11_s}, \tau_s, h_s, \frac{1}{\nu(\mathcal{G}^b_{CA_s})}, p_s)$;
36 $\mathcal{G}^b_s \leftarrow$ Map $\mathcal{G}_{I_s}$;
37 **if** $h_s = \frac{1}{\nu(\mathcal{G}^b_s)}$ **then**
38 $\chi^s_{s_s} = (\mathcal{G}^b_s, h_s, \tau_s, K_s, F_s)$;
39 **else**
40 // the mapping of $s_s$ is not feasible
go to Step 27 and choose a different (sub)set of system scenarios (possibly reverting to the worst-case scenario $s_{s_{wc}}$ as the single system scenario);
41 Create a LUT for runtime use (as explained in Section 4.3.3);

ecution time $e_{CA}$. If the control compute and actuate tasks are single actors $C$ and $A$ respectively, then $e_{CA} = e_C + e_A$. However, if the control compute and actuate tasks are parallelised (sub)graphs $\mathcal{G}_C$ and $\mathcal{G}_A$, we need to find the latency from the combined binding-aware graph after the $RepA$ and $Pipe$ transformations (**Step 6**). In this case, the latency is equal to the inverse throughput due to the $Pipe$ transformation, i.e., $e_{CA} = 1/\nu(\mathcal{G}^b_{CA_i})$. **Step 7** maps initial implementation-aware graph $\mathcal{G}_{11_i}$ to the given platform allocation to obtain the binding-aware graph $\mathcal{G}^b_{11_i}$. **Steps 8 and 9** compute the sensor-to-actuator delay $\tau_i$ and sampling period $h_i$ for $s_i$ from this binding-aware graph (as explained earlier in Eq. 4.2).

If pipelining is allowed, i.e., $p > 1$, then we go through an extra iteration of the timing-analysis loop (in Fig. 4.3) to compute the inter-frame dependence time for the scenario at hand, $f^i_d$, for the pipelined implementation (**Steps 10 - 14**). **Step 11** refines the initial implementation-aware graph for the scenario at hand with delay-awareness, pipe replication, camera-awareness, and workload-awareness through Algorithm 1 with the timing values computed in the previous steps. In order to compute the inter-frame dependence time, we then set the execution time of the camera actor to zero (Step 12) so that the inter-frame dependency is the throughput limiting factor in our graph. We map the refined graph $\mathcal{G}_{32_i}$ to obtain $\mathcal{G}^b_{32_i}$ and compute $f^i_d$ as the inverse throughput of $\mathcal{G}^b_{32_i}$ (Step 14). A point to note is that the actors camera, delay$_j$ and $T_j$ being added in the process are not mapped to the given platform allocation (while mapping in SDF3, we bind each of these actors to separate dummy processors). These actors are required to simulate time-triggering of tasks and ordering of pipes.

In **Steps 17 - 25**, we proceed with the timing analysis in Fig. 4.3 to compute the inter-frame dependence time $f_d$ for the IBC application as a whole and the constant effective sampling period $h_{eff}$ for a pipelined implementation (i.e. $p > 1$). Recall from Section 4.3.3 that we enforce a constant sampling period for the pipelined implementation to limit the design space to be explored, reduce runtime overhead, and facilitate controller design. The computation of $h_{eff}$ starts with determining the worst-case delay $\tau_{wc}$, worst-case period $h_{wc}$, and corresponding inter-frame dependence time $f_d$. We can then compute the maximum number of pipes feasible $p_{max}$ due to inter-frame dependencies and the maximum number of cores we require $n_{c_{max}}$ to realise $p_{max}$. We can then compute the smallest realisable sampling period $h_{min}$, after which we set $h_{eff}$ to the maximum of $h_{min}$ and $h_{wc}$.

**Step 18** determines the worst-case delay $\tau_{wc}$ and worst-case period $h_{wc}$. Because delay and sampling period are determined from a single pipe, the scenario with the largest delay also has the largest sampling period. Next, we compute the maximum inter-frame dependence time $f_d$ (**Step 19**) over all the workload scenarios $s_i$. The maximum (and not any other) inter-frame dependence time is considered for further analysis since the order of the workload scenario sequence at runtime is not known apriori. Because of inter-frame dependencies, not all frames can be used for sensing. With $n_s$ as computed from $f_d$ and $f_h$ as indicated in **Step 20**, $n_s - 1$ is the effective number of frames skipped between processing the arriving

camera frames due to the inter-frame dependencies. The maximum operation is required to avoid a corner case in the subsequent analysis when $f_d = 0$. $n_{f_{wc}}$ (**Step 21**) is the number of camera frames arriving within any worst-case sensor-to-actuator delay interval for the single pipe execution. The realisable maximal number of pipes $p_{max}$ captures the maximum number of pipes possible for our pipelined implementation considering the frames we have to skip due to inter-frame dependencies and the total number of frames arriving within the worst-case delay interval $n_{f_{wc}}$ (see **Step 22**). We can then compute the maximum number of cores required for realising our design choices of $n_c^{//}$ and $p_{max}$ during runtime implementation as $n_{c_{max}}$ (**Step 23**). For instance, if we allocate two cores per pipe for parallelism and we would like to have two pipes, then we need a maximum of four cores. $h_{min}$ is then the minimum realisable sampling period possible for the controller implementation considering the given choice of parameters; it can be computed as shown in **Step 24**. If more cores are allocated than the maximum number of cores required to realise our design choices, i.e., $n_c^{avl} \geq n_{c_{max}}$, then $h_{min}$ is limited only by the inter-frame dependencies, as captured by $n_s$. In this case, the SPADE implementation utilises a maximum of $n_{c_{max}}$ cores, as having more cores does not improve $h_{min}$ and, in effect, does not improve the control performance. However, if the resources we require to realise a sampling period of $n_s \times f_h$ are not allocated, i.e., $n_c^{avl} < n_{c_{max}}$, then $h_{min}$ has to be increased proportionally to the fraction $\frac{n_{c_{max}}}{n_c^{avl}}$. E.g., let $n_{c_{max}} = 4$, $n_s = 1$. If $n_c^{avl} \geq 4$, we can achieve the sampling period $h_{min} = f_h$. However, if $n_c^{avl} = 2$, we cannot realise $h_{min} = f_h$. In this case, we increase $h_{min}$ as many times as the fraction $\frac{4}{2}$, i.e., $h_{min}$ becomes $\frac{4}{2} n_s \times f_h = 2 f_h$. The effective realisable sampling period $h_{eff}$ is finally taken as the maximum of $h_{min}$ and $h_{wc}$ (**Step 25**).

**Steps 27 - 40** design controllers for the workload scenarios, identify the system scenarios $s_s$, derive the binding-aware graph $\mathcal{G}_s^b$ for $s_s$, check feasibility of the scenario definitions, and define the system configurations. These steps are illustrated in Fig. 4.3 using the blocks controller design, system-scenario identification, and system configurations. For a non-pipelined implementation, controllers are designed as explained in Section 4.5.3 and the system-scenario identification is done as explained in Section 4.3.2. For a pipelined or pipelined-parallelism implementation, controller design and system-scenario identification are explained in Sections 4.5.4 and 4.5.4, respectively. Recall that if we cannot guarantee the stability of the switched system being defined, our controller design reverts to a periodic worst-case-based design with a single worst-case system scenario. **Step 30** identifies this worst-case system scenario $s_{s_{wc}}$ as the scenario with the largest delay $\tau_{s_{wc}}$ (and hence also the largest period). If any of the identified system scenarios cannot be mapped onto the allocated resources in such a way that all timing requirements are met, then SPADE reverts to this worst-case scenario as the only system scenario as well (**Step 40**). **Step 31** computes the realisable number of pipes, i.e. the effective number of pipes in implementation, based on the worst-case timing analysis, controller design and scenario identification. $p_s$ is always less than or

equal to $p$.

For each of the identified system scenarios, **Steps 33 - 36** derive its binding-aware graph. We start from the initial implementation-aware graph of the contributing workload scenario with the maximum delay (**Step 33**). Then we identify the contributing workload scenario's binding-aware graph for the control compute and actuation tasks (**Step 34**). **Step 35** refines the initial implementation-aware graph with the updated timing information on delay $\tau_s$, period $h_s$, execution time for the control compute and actuate tasks ($e_{CA_s} = 1/\nu(\mathcal{G}^b_{CA_s})$ and the realisable number of pipes $p_s$ using Algorithm 1. Finally, we map the updated implementation-aware graph to the platform (see **Step 36**) and check if the control timing is realisable in the binding-aware graph (**Step 37**). Control timing is realisable if a feasible mapping exists for the binding-aware graph, and a feasible mapping implies that the inverse throughput of the $\mathcal{G}^b_s$ is equal to $h_s$. If a feasible mapping exists, we can define the system configuration $\chi^s_{s_s}$ (**Step 38**) for the system scenario $s_s$. If the mapping is infeasible, we need to choose a different subset of system scenarios and re-do the controller design as explained earlier ( **Step 40**).

Once the feasible system scenarios have successfully been identified, we create the LUT in **Step 41**, as explained in Section 4.3.3. A DSE (illustrated in Fig. 4.3) is performed if we want to explore which implementation choice gives the best control performance. In this paper, we consider a brute-force DSE by varying the inputs to Algorithm 2, and analysing the performance of the resulting system configurations.

### 4.5.3 SPADE control design and switching for non-pipelined implementation

This section explains the controller design, in particular, system augmentation and switching, for non-pipelined implementation ($p = 1$). Recall from Section 4.3.2 that we aggregate workload scenarios based on the camera frame rate to limit the number of switching scenarios. Controllers are designed for the aggregated workload scenarios and system scenarios are identified as the switching-stable aggregated workload scenarios as explained in Section 4.3.2.

The control timing parameters $\tau_i$ and $h_i$ for any scenario $s_i$ are computed during the SPADE analysis and design (see Steps 8 and 9 of Algorithm 2). The controller design for the non-pipelined implementation has already been explained in Section 2.2.2.

Switching due to workload variations and switching stability have been explained in Section 4.3.2. Having numerous switching scenarios often results in instability [125] and degrades control performance due to the non-smooth response associated with the switching overhead. For optimising control performance and stability, it is essential that we limit the number of switching scenarios through system-scenario identification (as explained in Section 4.3.2). For a non-pipelined implementation, switching results in both variable delay and vari-

able period. SPADE flow for the non-pipelined implementation does not have any restrictions on the value of $\tau_s$ we can have for a system scenario. $h_s$ can vary, but should always be a multiple of $f_h$ to align the start of the sensing task with the camera frame rate.

### 4.5.4 SPADE refinements for pipelined implementation

For a pipelined implementation, i.e., for $p > 1$, we have that $\tau_{wc} > h_{eff}$ where $\tau_{wc}$ is the worst-case sensor-to-actuator delay, and $0 < \tau_i \leq \tau_{wc}$. For the scope of this work, we enforce a constant sampling period $h_{eff}$ (which is a multiple of $f_h$) for the overall pipelined implementation. The constant sampling period means that we ensure a constant start of sensing and also, a constant actuation rate. The computation of $h_{eff}$ depends on the inter-frame dependencies, $\tau_{wc}$, $f_h$, $p$, $n_c^{//}$, and $n_c^{avl}$ (see Algorithm 2 for the precise details). This section explains the significance of inter-frame dependencies, switching due to image workload variations in pipelining, controller design, system-scenario identification, the need for implementation-aware matrices and how we compute control configuration for pipelined implementation.

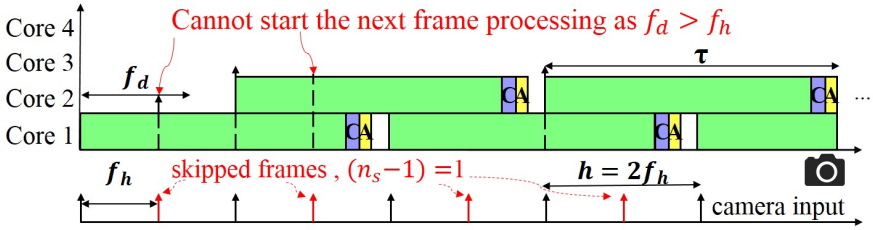



Figure 4.9: Illustration of inter-frame dependencies with $f_h < f_d \leq 2f_h$. (Adapted from Fig. 3.6, for readability.)

#### Inter-frame dependencies in a pipelined implementation

Pipelining is inherently limited by inter-frame dependencies, i.e., the data or algorithmic dependencies between consecutive frame processing, e.g., due to video coding [77] or visual tracking [120]. Considering inter-frame dependencies is crucial for a practical pipelined implementation. Inter-frame dependence time (denoted by $f_d$) can be quantified for the current image frame as the maximum time required to complete the processing of (parts of) the IBC algorithm the subsequent image frame processing depends on. Alternatively, $f_d$ is the minimum time required to wait between the start of processing consecutive image frames. Fig. 4.9 illustrates the impact of inter-frame dependence time on sampling period $h$. In a pipelined implementation, considering inter-frame dependencies means that

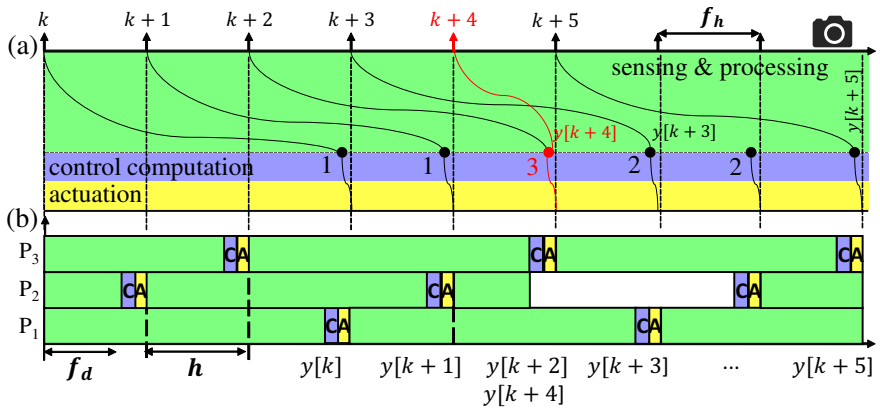



Figure 4.10: Illustration of switching due to workload variations in a multiprocessor pipelined implementation. (a) Logical delay diagram (adapted from Fig. 3.7) illustrating the cases explained in Sec 4.5.4 and (b) its corresponding Gantt chart (adapted from Fig. 3.3). The sample $k+4$ has a lower workload and thus the latest output measurement $y(k+4)$ is available within one $f_h$.

strictly $h_{eff} \geq f_d$. The number of frames that has to be skipped after processing every frame is $n_s - 1$ with $n_s$ computed as in Step 20 of Algorithm 2. This is illustrated in Fig. 4.9 where $n_s = 2$ and one frame is skipped after every frame processing.

The computation of $f_d$ is explained in Algorithm 2. The inter-frame dependencies are modelled using the *ifd* model transformation explained in Section 4.4. Computing $f_d$ helps to determine the effective image arrival period or the minimum possible sampling period $h_{min}$ we can have. Inter-frame dependencies mean that sometimes image frames have to be skipped for processing with respect to the given image arrival period $f_h$ and the sampling period $h$. Skipping a frame means that $h$ increases and thus degrades the control performance. In some cases, e.g., when the sensing uses video coding, inter-frame dependencies limit the effective camera frame rate and the video needs to be encoded/decoded at the effective rate $(1/h_{eff})$. In case the video encoding is closed-source and the video encoding cannot be done at a new rate, then sufficient resources need to be allocated first for decoding at the original camera frame rate and only the remaining resources can be utilised for the rest of the application. In this case, the video decoding is periodically executed at the camera frame rate and mapped first to the given platform allocation. The IBC application is then mapped to the remaining allocation.

For a non-pipelined implementation, the inter-frame dependencies can be ignored since the frames are processed in sequence.

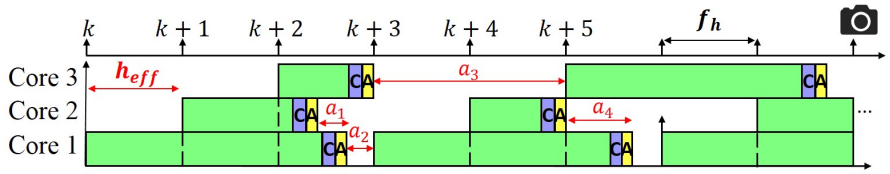



Figure 4.11: Illustration of the challenge with varying actuation rates $a_i$ due to variable $\tau'_i$.

### Switching in pipelined implementation

Switching in a multiprocessor pipelined IBC system implementation due to workload variations has not been explicitly explored in literature apart from our previous work explained in Chapter 3. When we do not consider workload variations, a pipelined implementation effectively results in a constant $\tau$ and $h$. Considering workload variations implies that we would have varying sensing delays, e.g. as illustrated in Fig. 4.10. Here, notice that the camera input frame at $k$+4 has a sensing delay of one frame ($\tau_1 = h$) due to lower image workload, and all other frames have a sensing delay of three frames ($\tau = 3h$). This scenario results in multiple sensing and image processing (S) tasks completing their execution at the same time. This means that multiple output measurements $y[k+2]$, and $y[k+4]$ are available for control computation task $C$ at the same time instance. For the scope of this work, the sampling period is kept constant and the switching happens due to variable delay.

Notice that by having just one frame with a lower workload, we can have the following three switching cases as illustrated in Fig. 4.10 (a): case 1) the new measurement is available with the same sensing delay as in the previous step; case 2) the sensing delay is increased compared to the previous step as the latest measurement is not available. Older past measurements may be available during this time (e.g. $y[k+3]$ becomes available one period after $y[k+4]$). However, they are used only to update the state estimates and not directly for control input computation; case 3) the sensing delay is reduced by one or more steps: when multiple pipes finish processing a corresponding sequence of frames, both the latest measurement(s) along with the past measurements are now available. The past measurements are used to update the state estimates, and the latest measurement is used to compute the control input.

Thus, the main challenge for the pipelined IBC system design in order to maximise performance, i.e. QoC, is to effectively use the sensor measurements as early as possible for control computation without any unnecessary idling and to estimate the system state when there are no sensor measurements available. Modelling this behaviour is far from trivial. This problem was explored with respect to long network delays in [79]. We leverage these results in our design.

**Control design and system-scenario identification**

We consider a pipelined implementation with workload scenarios $s_i$ having $(\tau_i,\ h_i)$ as timing parameters. It is possible to have a varying $h_i$ similar to the non-pipelined implementation. However, proving stability guarantees then becomes challenging and as such is not explored in this work. For the scope of this work, we assume a constant period $h_{eff}$ for all scenarios in the pipelined implementation. How we compute $h_{eff}$ was explained earlier in Algorithm 2. For brevity, $h = h_{eff}$ in the rest of this section.

For a workload scenario $s_i$, we can represent $\tau_i$ based on [100] as

$$\tau_i = (n_{f_i} - 1)h + \tau'_i,\ \text{where } 0 < \tau'_i \leq h,\ n_{f_i} = \left\lceil \frac{\tau_i}{h} \right\rceil. \tag{4.5}$$



This representation divides the delay $\tau_i$ into $n_{f_i}$ regions in the time domain. This results in $(n_{f_i} - 1)$ regions of $h$ and the left-over delay $\tau'_i$ for $\tau_i$. $n_{f_i}$ is the number of frames arriving in one delay period for the scenario at hand. The above $n_{f_i}$ computation assumes the practical situation where $\tau_i > 0$. In case one wants to consider $\tau_i = 0$, $n_{f_i} = max\left(\left\lceil \frac{\tau_i}{h} \right\rceil, 1\right)$.

Next, we enforce a constant actuation rate since a varying actuation rate results in undesired behaviour as illustrated in Fig. 4.11, even in the case of a pipelined implementation with a constant sampling period. Fig. 4.11 executes a scenario sequence $(s_3 s_2 s_1 s_3 s_1 s_3)^\omega$ with sampling period $h_{eff} = f_h$. The scenario $s_1$ has the best-case delay $\tau_1 = f_h$, $s_2$ has a delay $\tau_2 = 1.2 f_h$, and $s_3$ has a delay $\tau_3 = 2.5 f_h$. If we now apply Eq. 4.5, we get a varying $\tau'_i$ and the Gantt chart as illustrated in Fig. 4.11. Notice that the actuation rates illustrated by the $a_i$ are not periodic anymore and we have ordering issues as well with respect to the actuation task. Further, guaranteeing controller stability for this case is challenging.

We mitigate varying actuation rates by defining $\tau' = \max_i(\tau'_i)$ and then designing controllers with $\tau_i = (n_{f_i} - 1)h + \tau'$ and $h = h_{eff}$ for workload scenarios $s_i$. A constant $\tau'$ over all scenarios is required to maintain a constant actuation rate between switching scenarios in a pipelined implementation. Notice that by defining and fixing $\tau'$ we are aggregating workload scenarios with the same $n_{f_i}$ (see Eq. 4.5). The controllers are designed for these aggregated workload scenarios. Also, the design space for system-scenario identification is narrowed down by the workload scenario aggregation.

To design the controllers, Eq. 4.1 can then be reformulated as follows [100]:

$$\begin{aligned} x[k+1] = A_{s_i} x[k] + B'_{0,s_i} u[k - (n_{f_i} - 1)] \\ + B'_{1,s_i} u[k - n_{f_i}], \end{aligned} \tag{4.6}$$

where $A_{s_i}$, $B'_{0,s_i}$ and $B'_{1,s_i}$ are given by replacing $\tau_i$ by $\tau'$ and $h_i$ by $h$ in Eq. 2.3. It is interesting to note that the matrices $A_{s_i}$, $B'_{0,s_i}$ and $B'_{1,s_i}$ are identical for all the scenarios due to this formulation. However, Eq. 4.6 is still different for different

aggregated scenarios due to varying $n_{f_i}$. We leverage the identical matrices during the runtime implementation as we only have to store a few key matrices as explained in Section 4.5.4.

Next, we define new augmented system states $z'[k] = \begin{bmatrix} x[k] & u[k-(n_{f_i}-1)] & \cdots & u[k-2] & u[k-1] \end{bmatrix}^T$ to obtain a higher-order augmented system as follows:

$$z'[k+1] = A'_{s_i} z'[k] + B'_{s_i} u[k],\ y[k] = C' z'[k] + D_c u[k],$$

$$A'_{s_i} = \begin{bmatrix} A_{s_i} & B'_{1,s_i} & B'_{0,s_i} & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix},\ B'_{s_i} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ I \end{bmatrix},$$

$$C' = \begin{bmatrix} C_c & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{4.7}$$

Controllers are then designed for this higher-order augmented system, as explained in Section 4.3.2. The stability criterion for the switched pipelined implementation is similar to the problem for long network delays [28, 79] and as such is not explained here.

For the pipelined implementation, we identify system scenarios using the following steps, where the first two steps are done as part of the controller design: i) classify the workload scenarios $s_i$ with the same $n_{f_i} = \lceil \frac{\tau_i}{h} \rceil$ into a maximum of $\lceil \frac{\tau_{wc}}{f_h} \rceil$ aggregated workload scenarios; ii) for each aggregated workload scenario, design a controller with $\tau_i = (n_{f_i} - 1)h + \tau'$ and $h = h_{eff}$; iii) check for control stability for the switched system with the identified aggregated workload scenarios. If the switched system is unstable, find an aggregation of workload scenarios for which the switched system is stable through an exhaustive search. If multiple scenario sets provide a stable system, we choose the set with the highest cardinality and shortest average sensor-to-actuator delay; iv) the identified aggregated workload scenarios that result in a stable switched system are the system scenarios $s_s$ with $\tau_s = (n_{f_s} - 1)h + \tau'$ and $h = h_{eff}$.

#### Implementation-aware control matrices and control configurations

When we allow switching for pipelined implementation, the challenge is the varying dimensions of matrices in Eq. 4.7 for varying delays due to workload variations. The varying dimensions affect the runtime computation of $u[k]$ (see Eq. 4.4), where the matrix $K_i$ needs to be multiplied with $z[k]$ at every time-step. This challenge also occurs when some of the system states are estimated and not directly obtained from sensor measurements. For the LKAS, only the third state is computed from the sensor and the other states are estimated using the system model

(see Eq. 4.3). In this case, $z[k+1]$ (see Eq. 4.3) needs to be computed at every time step.

We tackle the challenge of varying dimensions of matrices by unifying/normalising the dimensions of matrices considering the worst-case delay $\tau_{wc}$ over all system scenarios $s_s$. The matrices for the worst-case delay scenario $s_{wc}$ annotated with $(h, \tau_{wc})$ are the same as in Eq. 4.7. Let $n$ be the order of the square matrix $A_{s_{wc}}$ and $n_{f_{wc}} = \lceil \frac{\tau_{wc}}{h} \rceil$. For each system scenario $s_s$, with $\tau_s > h$, $n_{f_s} = \lceil \frac{\tau_s}{h} \rceil$, the system matrices are given below. For brevity, let $n_{f_{wc}} = n_f$ and $n$ is the order of matrix $A_{s_s}$. It is interesting to note that $A_{s_s} = A_{s_{wc}}$ when the period $h$ is constant for the system scenarios $s_s$. The order of the square matrix $A'_{s_s}$ is $(n + n_f)$.



$$A'_{s_s} = \begin{bmatrix} A_{s_s} & 0_{(n_f-1)\times n} & 0_{1\times n} \\ 0_{n\times(n_f-n_{f_s})} & & \\ B'_{1,s_s} & 0_{(n_f-1)\times 1} & 0_{1\times 1} \\ B'_{0,s_s} & & \\ 0_{n\times(n_{f_s}-2)} & I_{(n_f-1)\times(n_f-1)} & 0_{1\times(n_f-1)} \end{bmatrix}^T,$$

$$B'_{s_s} = \begin{bmatrix} 0_{(n+n_f-1)\times 1} & I_{1\times 1} \end{bmatrix}^T, \; C' = \begin{bmatrix} C_c & 0_{1\times n_f} \end{bmatrix}.$$

Further, in a pipelined implementation due to workload variations we could have a scenario $s_s$ with $\tau_s \leq h$ (see for instance iteration $[k+4]$ in Fig. 4.10). We derive the matrices for such scenarios as follows. Also, since $\tau_s \leq h$, $n_{f_s} = 1$.

$$A'_{s_s} = \begin{bmatrix} A_{s_s} & 0_{n\times(n_f-n_{f_s})} & B'_{1,s_s} \\ 0_{(n_f-1)\times n} & 0_{(n_f-1)\times 1} & I_{(n_f-1)\times(n_f-1)} \\ 0_{1\times n} & 0_{1\times 1} & 0_{1\times(n_f-1)} \end{bmatrix},$$

$$B'_{s_s} = \begin{bmatrix} B'_{0,s_s} & 0_{(n_f-1)\times 1} & I_{1\times 1} \end{bmatrix}^T,$$

$$C' = \begin{bmatrix} C_c & 0_{1\times n_f} \end{bmatrix}.$$

The matrices $A_{s_s}$, $B'_{0,s_s}$ and $B'_{1,s_s}$ are obtained for scenario $s_s$ as explained in Eq. 4.7. We can apply standard control design techniques for these implementation-aware system models. State-feedback and feed-forward controllers can be designed as shown in Eq. 4.4 for the system scenarios to obtain $K_s$ and $F_s$. The control configurations are then $\chi^c_{s_s} = (h_{eff}, \tau_s, K_s, F_s)$.

Note that when storing matrices for a pipelined implementation, we only need to store $K_s$ and $F_s$ for each scenario $s_s$, $A_{s_{wc}}$, $B'_{0,s_s}$, $B'_{1,s_s}$, and $C_c$ for the constant period $h$ and $\tau'$. We only need to store one $A_{s_{wc}}$, $B'_{0,s_s}$, $B'_{1,s_s}$, and $C_c$ matrix as the period and actuation rate are constant.

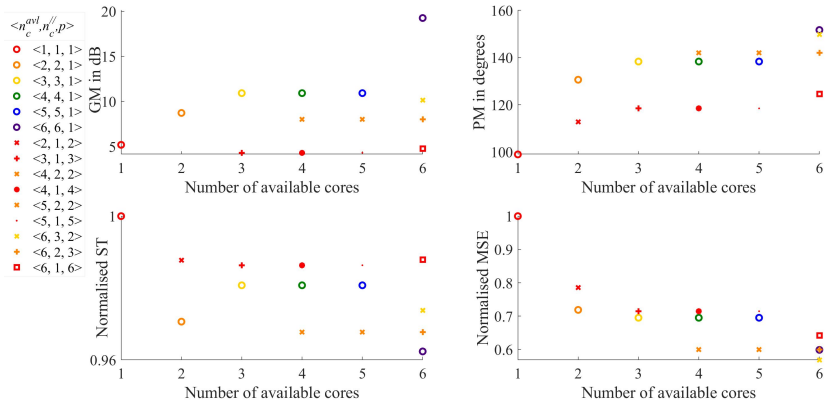



Figure 4.12: Pareto-plot for simulation - QoC vs given platform allocation, i.e., $n_c^{avl}$. Here, we consider the SADF in Fig. 4.4 (a) with $z = 6$ (worst-case workload). The MSE and ST are normalised with respect to the maximum (worst-case) value. The legend denotes $< n_c^{avl},\ n_c^{//},\ p >$. Higher GM and phase margin (PM), and lower MSE and ST are better.

## 4.6 Experimental results and discussion

This section explores the SPADE experimental results with a focus on DSE. The DSE is used for identifying the degrees of parallelism and pipelining for the pipelined parallelism implementation. A DSE is required since the parallelisation is limited by the degree of application parallelism quantified using $n_c^{//}$ and $n_c^{avl}$, and the maximum number of active pipes we can have is limited by $f_d$, $f_h$, $n_c^{//}$ and $n_c^{avl}$. After exploring the DSE results for the running example, we discuss a few observations for the SPADE flow and compare the state-of-the-art methods with our proposed SPADE flow for pipelined parallelism. The simulations in this section consider the LKAS case study introduced in Section 1.7. The LKAS is modelled using the IBC graph given in Fig. 4.4 (a) with the worst-case workload scenario having $z = 6$ and the execution times of the actors as explained in Section 4.3.1. The platform we consider is the predictable COMPSOC platform (as explained in Section 1.3.1) with $n_c^{avl}$ as an input to the SPADE flow.

### 4.6.1 Design-space exploration (DSE)

The SPADE flow has been explored till now with a fixed number of pipes $p$ and a fixed number of cores per pipe for parallelism $n_c^{//}$. A DSE with design parameters $f_h,\ p,\ n_c^{//},\ n_c^{avl}$ is needed to identify the Pareto-optimal implementation choice, i.e., the degree of pipelining and the degree of parallelism. Note that $p$ quantifies the degree of pipelining, and $n_c^{//}$ quantifies the degree of application parallelism

for an implementation choice. Typically, $f_h$ and $n_c^{avl}$ are given or fixed. We only vary these two parameters if the goal is to identify the optimal frame rate or minimise resource usage.

We use a brute force method to identify the possible implementation choices $< n_c^{avl},\ n_c^{//}, p >$ over the design space. For each implementation choice, i.e., with fixed design parameters, we compute its GM and PM (see Section 4.2.2) for the worst-case system scenario $s_{s_{wc}}$ (see Step 30 in Algorithm 2). GM and PM are analytical metrics and can be computed easily from the discretized control system model in Eq. 4.3 for $s_{s_{wc}}$. We suggest using GM and PM to prune the design space to be explored for multiple implementation choices, as the MSE and ST QoC metrics that we ultimately want to optimise are simulation-based and obtaining them is compute-intensive. A performance comparison for different implementation choices using MSE and ST is moreover unfair unless we can do exhaustive simulation considering different initial conditions and environments (e.g., weather conditions, as illustrated in our previous work [36]).



Thus, the Pareto-optimal implementation choice for a given platform allocation $n_c^{avl}$ is chosen as the one with the highest GM and PM. If GM and PM are incomparable for multiple implementation choices for a platform allocation, in the sense that one implementation choice has a higher GM and the other one a higher PM, then we consider both these choices as part of the Pareto front. The Pareto-optimal implementation choices are further analysed using MSE and ST. To do so, controllers are designed for the Pareto-optimal implementation choices, and optimal system scenarios considering workload variations are identified based on control performance metrics MSE and ST.

As a proof-of-concept, we performed DSE with $f_h = \frac{1}{60}$ and given (maximum) platform allocation of six processing cores, i.e., $n_c^{avl} = 6$, considering the IBC graph in Fig. 4.4 (a). The QoC over various design points is illustrated in Fig. 4.12. For the purpose of validation, we explored more implementation choices through simulation than only the Pareto-optimal ones as described above. The results show that the implementation choices with higher gain and phase margins indeed have better control performance (MSE and ST) in the simulations, confirming that GM and PM can be used to prune the design space. The only anomaly is the normalised ST for the case $< 2,2,1 >$ for two cores which is better than the ST for the configurations with a higher number of cores. We observe that this is due to our simulations proceeding at the rate of the sampling period. E.g., the sampling period for both the cases $< 2,2,1 >$ and $< 3,3,1 >$ is the same (0.0667 ms), but their sensor-to-actuator delays are 0.065 ms and 0.055 ms, respectively. When we proceed with the simulations at the rate of the sampling period, the slight differences in the settling time are due to approximations in the model fitting.

In terms of optimising QoC for the running LKAS example, an important first observation from the DSE results is that all configurations that exploit parallelism and/or pipelining improve QoC over the fully sequential implementation $< 1,1,1 >$. Our simulations moreover show that, typically, for the optimal QoC,

we should parallelise as much as possible (increasing $n_c^{//}$) and then pipeline (increasing $p$). For example, in Fig. 4.12, let us consider the cases of $n_c^{avl} = 3$ and $n_c^{avl} = 6$. Among the configurations with $n_c^{avl} = 3$, we notice that the configuration $< 3,3,1 >$ has the highest GM and PM and the best control performance. The normalised MSE is similar for both $< 3,3,1 >$ and $< 3,1,3 >$. There is a visible improvement in the normalised ST for the first of these two configurations that maximises parallelism compared to the second that maximises pipelining. Similarly, when $n_c^{avl} = 6$, we notice that configuration $< 6,6,1 >$ has the highest GM and PM and best normalised settling time. The normalised MSE is similar to the cases $< 6,3,2 >$ and $< 6,2,3 >$. We observe that this is due to having the same sampling period for all these cases. The trend that parallelisation should be prioritised over pipelining is true for other platform allocations.

Let us now consider the cases of $n_c^{avl} = 4$ and $n_c^{avl} = 5$. It is interesting to notice that the control performance for both these cases is identical, and the performance does not improve by allocating one more core when $n_c^{avl} = 4$. This is due to identical control timing parameters, delay and period, when $n_c^{avl} = 4$ and $n_c^{avl} = 5$. It is also interesting to note that the fully parallelisable implementations $< 4,4,1 >$ and $< 5,5,1 >$ do not have the best performance for both these cases. This is because of identical delay (= 55 ms) and period (= 66.7 ms) when $n_c^{//} = n_c^{avl}$ and $p = 1$, for the cases with $n_c^{avl} = 3,4,5$. This happens because when we distribute six ROIs over 3, 4 or 5 cores, there are always two sequential ROI executions needed on (at least) one core, meaning that the effective delay does not change. This means that we already have a fully parallelisable implementation with $n_c^{avl} = 3$. When we have a fully parallelisable implementation and still have more cores available, we can pipeline. E.g., the cases with the best performance for $n_c^{avl} = 4,5$ are $< 4,2,2 >$ and $< 5,2,2 >$. These cases have delay and period equal to 65 ms and 33.3 ms, respectively. We observe that considering parallelism and pipelining integrally has better performance than considering only one of the two options.

The main conclusion of the DSE is that we should *parallelise as much as possible and then pipeline while taking into account the delay and period.* It is important to note that increasing the number of available cores does not necessarily improve the control performance. This is due to several factors. The most prominent factors, the camera frame rate and the inter-frame dependencies, have been discussed earlier. Our results also show the significance of GM and PM in pruning the design space for exploration. A higher GM and PM typically imply a better control performance (MSE and ST). The GM and PM are computed analytically, whereas the MSE and ST can only be computed through simulations. For SPADE, we use this knowledge to prune the design space.

### 4.6.2 Is higher frame rate always better?

Our observation with respect to the frame rate is that having a higher frame rate is not always better. Having a higher frame rate means processing the arriving frames at a higher rate, which is compute-intensive, and may not always improve the con-

trol performance (as shown in Fig. 4.13). We observe that the control performance is similar for 60 fps and 120 fps for all the cases. The control performance is the worst for 30 fps when we have only one available processing core. A slight degradation in control performance is noticed for 30 fps when considering 2-5 available processing cores. If we have six available processing cores, all the frame rates have similar performance. The control performance is dependent mainly on the $\tau_s$ and $h_s$ of the system scenarios. If improving the frame rate does not effectively decrease the average delay and/or period, it becomes an overhead to do so and wastes compute resources in the given platform.



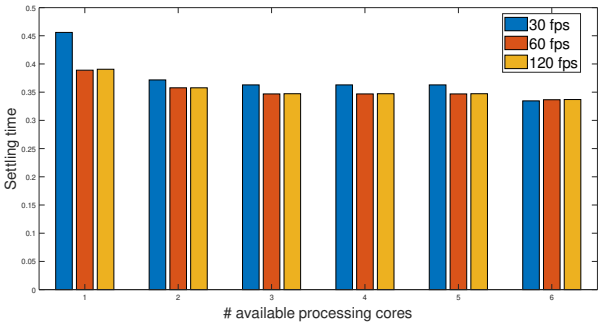



Figure 4.13: $n_c^{avl}$ vs settling time (in s) considering different frame rates for the SADF in Fig. 4.4 (a) with $z = 6$ (workload) and $p = 1$.

### 4.6.3 Impact of inter-frame dependencies

In a pipelined implementation, the inter-frame dependencies have a significant impact on the maximum number of pipes we can realise $p_s$ (see Step 31 of Algorithm 2). The impact of the inter-frame dependence time $f_d$ on $p_s$ is illustrated in Fig. 4.14 using an example. In this example, we allocate sufficiently many cores $n_c^{avl} = 12$ since $n_{c_{max}} = 12$ (see Step 23 in Algorithm 2) for a camera frame rate of 120 fps, considering a worst-case delay of $\tau_{wc} = 95$ ms, and $n_c^{//} = 1$.

If $f_d = 0$, the maximum number of realisable pipes we can have when the camera frame rates are 30, 60 or 120 fps are 3, 6 or 12, respectively. E.g., if the camera frame rate is 120 fps, then we can have a maximum of $\lceil \tau_{wc}/f_h \rceil = \lceil 0.095 \times 120 \rceil = 12$ pipes. Note that, as $f_d$ increases, the number of realisable pipes reduces exponentially. E.g., if $f_h < f_d \leq 2f_h$, then the number of frames to skip after processing each camera frame is one. Now for the camera frame of 120 fps, skipping one frame after every frame processing means that the realisable number of pipes will become 6 (see the interval $f_d = (0.0083, 0.01667]$ in Fig. 4.14). A higher $f_d$ implies a smaller number of realisable pipes. Also, as $f_d$ increases beyond a certain point, pipelining is no longer feasible (when $p_s = 1$). E.g., in Fig. 4.14, $p_s = 1$ in the intervals $f_d = (0.067, 0.100]$ for 30 fps, $f_d = (0.083, 0.100]$ for 60 fps, and $f_d = (0.092, 0.100]$ for 120 fps.

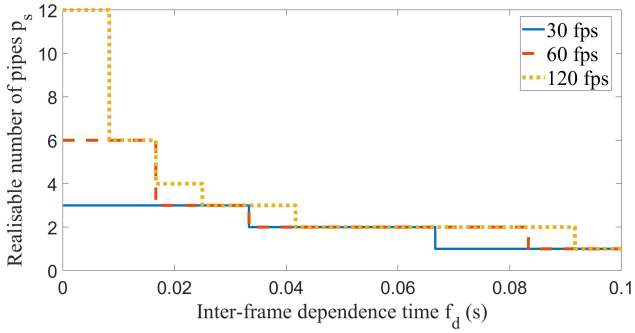



Figure 4.14: Impact of $f_d$ on maximum number of realisable pipes $p_s$. In this example, we consider $\tau_{wc} = 95$ ms, $n_c^{//} = 1$ and $n_c^{avl} = 12$.



### 4.6.4 Impact of system-scenario switching on control performance

It is possible for switching to degrade the control performance compared to the periodic worst-case-based design (presented in Fig. 4.12). However, we choose system scenarios such that the control performance improves even if we have switching at runtime. Further, we discard the combinations of aggregated workload scenarios that degrade control performance compared to the single worst-case workload scenario $s_{wc}$ during system-scenario identification (see Section 4.3.2). We illustrate this observation using Fig. 4.15 where we consider the SADF in Fig. 4.4 having three system scenarios $s_1$, $s_2$ and $s_{wc}$ with 1, 3 and 6 RoI respectively. $s_1$ is the best-case workload scenario with the smallest delay and $s_{wc}$ is the worst-case scenario with the worst-case delay. Notice that, in all switching cases, the control performance settles faster than the periodic worst-case based design (denoted by $(s_{wc})^{\omega}$ in Fig. 4.15). The trend is similar for different camera frame rates as well. We observe that the control performance is better if the frequently occurring workload scenario is closer to the best case than the worst case. E.g., in Fig. 4.15, the scenario sequence $(s_1^{10} s_{wc})^{\omega}$ has $s_1$ as the frequently occurring scenario, which leads to a better performance than $(s_2^{10} s_{wc})^{\omega}$ with $s_2$ as the most frequently occurring scenario. Further, we observe that the control performance of the scenario sequences cluster towards the performance of its most frequently occurring scenario, as is observed in Fig. 4.15 with distinct colour regions (where the blue and green scenarios form one cluster).

### 4.6.5 Comparison with the state-of-the-art

The DSE, as explained in Section 4.6.1 and illustrated in Fig. 4.12, already shows quantitatively that integrally considering the combination of parallelism and pipelining in multiprocessor IBC design outperforms the state-of-the-art approaches in which only one of the two options is considered. The integral consid-

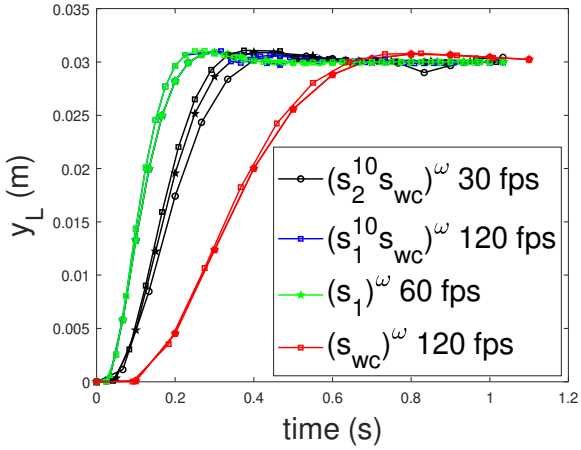



Figure 4.15: Impact of system scenarios switching due to workload variations on $y_L$ with $n_c^{avl} = 1$. Note that the legends mention representative scenario sequences; the plot contains more scenario sequences than the ones mentioned in the legends. The markers denote the frame rate and colours denote the scenario sequences.

eration of parallelism and pipelining differentiates SPADE from any earlier work, as explained in Section 1.4. In this subsection, we compare the proposed SPADE method qualitatively on several other relevant criteria with state-of-the-art multiprocessor IBC design techniques in Table 4.1. For brevity, we only compare with multiprocessor IBC system implementations and not with traditional sequential control design techniques based on the worst-case sensing delay, as it has already been shown in Chapter 2 and [46] that multiprocessor implementations are beneficial for optimising control performance. The multiprocessor implementations can be classified into pipelined [69], [112] with constant delay, pipelined with variable delay [82] and pipelined with workload scenarios (Chapter 3), and sequential implementation with parallelisable sensing that is explained in Chapter 2. The camera frame rate, however, is not explicitly considered in [69].

The proposed approach is advantageous to other multiprocessor IBC system implementations with respect to: 1) coverage of the design space considering both pipelining and parallelisation; 2) considering inter-frame dependencies and resource sharing among pipes; and 3) not imposing any restrictions on $\tau$, thereby enabling shorter $\tau$ and $h$ compared to other approaches.

## 4.7 SPADE adaptation for an industrial platform

We have explained the SPADE flow until now assuming that the timing of the implementation is predictable. However, for an industrial platform, it is difficult to

Table 4.1: Comparing the proposed SPADE approach with the state-of-the-art multiprocessor IBC system implementations

| Criteria | SPADE approach | Pipelined | | Chapter 2 |
|---|---|---|---|---|
| | | [69], [112], [82] | Chapter 3 | |
| Inter-frame dependencies | explicitly considered | not considered | considered | independent |
| System non-linearities | control-design dependent | not considered | explicitly considered | not considered in Chapter 2; control-design dependent |
| Constraints on variables | control-design dependent | cannot be imposed | can be strictly imposed | not considered in Chapter 2; control-design dependent |
| Control computation time | low | low - medium | high | low |
| Runtime overhead due to reconfiguration | explicitly considered as a time cost in the SADF model | not considered [69] [112]; not explicitly considered [82] | explicitly considered | explicitly considered |
| Algorithm | white/gray/black box | white/gray/black box | white/gray/black box | white/gray box |
| Parallelisation potential | explicitly taken into account | independent | independent | should be high |
| Workload variations | explicitly considered in design | not considered [69] [112]; indirectly considered [82] | explicitly considered in design | explicitly considered |
| Platform | can be adapted for all | suitable for homogeneous [69] [112]; can be adapted for all [82] | suitable for homogeneous | directly applicable for all |
| Resource sharing between pipes | explicitly considered during mapping | not considered | not considered | not applicable |
| Restrictions on sampling period $h$[1] | strictly periodic for pipelined, $h < \tau_{wc}$; switched for non-pipelined; | strictly periodic; $h < \tau_{wc}$ | strictly periodic; $h < \tau_{wc}$ | switching possible |
| Restrictions on sensor-to-actuator delay $\tau$ | none | strictly $\tau_{wc} > h$; in [112], $\tau$ is strictly a multiple of $h$ | strictly[2] $\tau_{wc} > h$ | $\tau_{wc} \leq h$ |

$\tau_{wc}$: worst-case delay; [1] If camera frame arrival period $f_h$ is considered, always $h$ is a multiple of $f_h$; [2] if $\tau_{wc} \leq h$ design reverts to non-pipelined.

predict the timing analytically (as users have restricted freedom in scheduling due to caching, resource sharing, etc.), and any computed timing is pessimistic. This section shows how we can apply the SPADE flow even when it is difficult to give tight predictable timing guarantees. As in Chapter 2, the idea is that we use the frequently occurring task execution times instead of the pessimistic worst-case execution time (WCET) estimates to obtain temporal bounds.

Chapter 2 explored this idea for a non-pipelined parallelisable implementation. In this chapter, we show that the SPADE adaptation for an industrial platform is relevant for pipelined parallelism as well. The assumption for implementation is that it is possible to time-trigger tasks either through polling or through interrupt timers in the industrial platform. We provide validation through a HIL simulation, showing that we can guarantee control stability for the SPADE design outcomes,

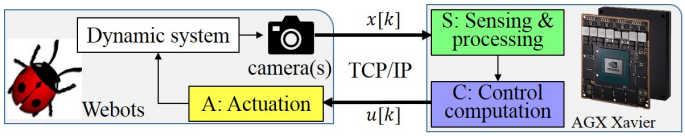



Figure 4.16: HiL setting overview with NVIDIA AGX Xavier.

despite the fact that the execution-time estimates for tasks used in the SPADE flow are not conservative.

### 4.7.1 HiL setting for our case study

Fig. 4.16 illustrates our HiL validation setup for LKAS adapted from [87]. It simulates a vehicle with a top look-ahead camera using the Webots [84] physics simulator engine and interacts with an NVIDIA AGX Xavier platform using the transmission control protocol/internet protocol (TCP/IP) protocol. The simulator works in a server-client configuration. Webots acts as the server while the NVIDIA platform acts as the client. The server (Webots) progresses simulation in full synchronisation with the client (NVIDIA AGX Xavier) [47]. At each simulation step, the camera sensor simulated in Webots generates a raw image containing state information $x[k]$, that is fed to the NVIDIA platform. It executes the sensing ($S$) and control ($C$) tasks to generate control input $u[k]$, which is communicated back to Webots for actuation. After actuation, the simulation progresses to the next step.

For our evaluation, the camera sensor in Webots is modelled based on the AR1335 complementary metal–oxide semiconductor (CMOS) digital image sensor [6] and is set to a resolution of 720p[2]. The camera frame rate is varied between 30 fps, 60 fps, and 120 fps, depending on the sampling period of the controller. The actuation dynamics are modelled based on [107]. A lane width of 3.25 m is considered, as per standard road-safety guidelines. The vehicle is initially positioned with a fixed bias of 15 cm from the lane centre to test the control performance. The Webots simulation step is set to 1 ms, while the vehicle speed is set to 50 km/hr.

### 4.7.2 Platform graph

We proceed to explain the abstraction we make for the platform graph in Fig. 4.3. The NVIDIA AGX Xavier platform has a Carmel central processing unit (CPU) complex with eight cores and a Volta GPU with 512 CUDA cores and 64 Tensor cores, as shown in Fig. 1.7. Modelling all the GPU cores separately may lead to state-space explosion and is inefficient for the dataflow timing and mapping analysis. Also, the proprietary GPU scheduler is closed-source and needs to be accessed

[2]State-of-the-art lane detection algorithms [98] operate on low-resolution images. So, we perform our evaluation using downscaled (512×256) sensor images. Our approach is also effective for high-res images.

through the CPU. The execution times are difficult to predict for the tasks mapped to the GPU when there are other shared tasks on the GPU. Therefore, we abstract and combine the execution times of the tasks mapped to the GPU along with the execution time of the CPU task that accesses it. Thus, the platform allocation is defined based on the total number of CPU cores available $n_c^{avl}$. For the NVIDIA AGX Xavier platform, we have a maximum of eight CPU cores, i.e., $n_c^{avl} = 8$.

### 4.7.3 IBC graph

Next, we explain the IBC graph of Fig. 4.3 used in this experiment and how we populate it with the profiling information. We model the IBC sensing algorithm using the IBC graph illustrated in Fig. 4.17. This graph structure is for the approximate setting 'S3' of the IBC system presented later in Chapter 6. The execution time numbers for the actors in the graph and the rates in the channels are obtained by mapping and profiling the IBC application on the NVIDIA AGX Xavier platform. We perform a model-fitting using the profiled timing information to update the execution time of the actors for each workload scenario. The resulting IBC SADF model is then one of the inputs to the SPADE flow explained in Algorithm 2.



For profiling, around 100 images are identified with varying image workloads. We execute each stage in the LKAS 100 times for every image in the dataset to reduce sensitivity to access locality. This helps to characterise the profiling information as a PERT distribution [1] for the latency of an iteration of the graph. Workload scenarios are classified based on the resulting PERT distribution by identifying regions in the distribution based on occurrence frequency. This results in scenario graphs with the same graph structure and channel rates, but different actor execution times. The workload scenarios for this setup are not based on RoI, as in our running example. For the worst-case scenario, we use the worst-case profiling numbers for the execution times of the actors. Other workload scenarios use the best-case, first quartile, median and third quartile profiling data. As an example, the model parameters for the workload scenario using third quartile profiling data are $e_{dm} = 10.3$ ms, $e_{dn} = 4$ ms, $e_1 = 0.3$ ms, $e_2 = 4.65$ ms, $e_d = 5$ ms, $e_p = 1.4$ ms, $e_m = 0.16$ ms (see Fig. 4.17), $e_C = 0.016$ ms, and $e_A = 0.5$ ms (assuming single actors Cand Afor the control compute and actuation tasks). Workload scenarios may also be classified based on other parameters, like RoI in the running example. Our previous work [91] details the case where the workload scenarios are classified based on different operating modes of an application due to environmental conditions. Here, we use the observed overall latency of the sensing, because it is difficult to predict execution times on the considered platform based on other parameters.

### 4.7.4 SPADE flow, DSE and HiL validation

Once we have the IBC SADF model with the profiling information for every workload scenario, the rest of the steps in the SPADE flow follows Algorithm 2. A DSE

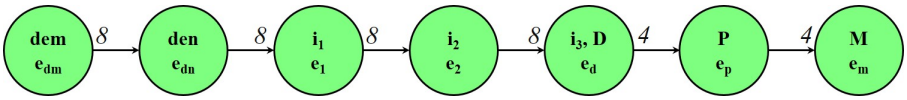



Figure 4.17: LKAS IBC graph of the sensing algorithm implementation derived from [35]. The actors are: dem - demosaicing; den - denoising; $i_{1,2,3}$ - abstract colour mapping and white balancing, tone mapping, and compression; and D-P-M model the lane detection, processing and merging tasks. The output is the lateral deviation $y_L$.



is performed for different implementation choices $n_c^{//}$ and $p$ (as explained in Section 4.6.1). A DSE involves analytical computation of GM and PM (in Matlab) for the different implementation choices (results are shown in Fig. 4.18). We consider three given platform allocations - single-core, four cores and eight cores, i.e., $n_c^{avl} = 1,4,8$. The implementation choices with the highest GM and PM are Pareto-optimal.

The results in Fig. 4.18 show that parallelising as much as possible gives the best result, i.e. the configurations $< 4,4,1 >$ and $< 8,8,1 >$ have the best GM and PM. The configurations $< 4,4,2 >$ and $< 8,4,2 >$ have similar GM and PM and can be considered for further analysis. The results confirm the earlier conclusion of Section 4.6.1 that parallelisation should be prioritised over pipelining. In this case, pipelining does not have any added value ($p = 1$ in both Pareto-optimal configurations), because the LKAS implementation is highly parallelisable. For applications with a lower degree of parallelism, e.g., when maximum parallelism is achieved with 4 cores and $n_c^{avl} = 8$, this would likely not be the case. In such a case, we can use the additional four cores for an additional pipe, implying that configuration $< 8,4,2 >$ would have been ideal.

Next, we validate the Pareto-optimal implementation choices in the HIL setting of Section 4.7.1 for control stability, considering both parallelism and pipelining together. Recall that the SPADE flow has a built-in stability check for the system configurations considering the initial system model. However, the actual LKAS implementation has to cater to different environment conditions, noise levels, and model uncertainties. HIL validation is often used to simulate the designed system configurations under real-life environment conditions. Therefore, we also perform a HIL validation experiment. From the experiment, we observe that stability of the implementation choices of the closed-loop system is confirmed. We can moreover verify that the order of the control system performance obtained in HIL conforms to the GM and PM predictions. As explained, we rely on analytical metrics GM and PM to select our design points. The MSE and ST comparisons are omitted as performing an exhaustive, fair simulation for even a single design point is too compute intensive. But the DSE performed with the SPADE flow and the HIL validation of the results show that SPADE can be adopted for industrial platforms.

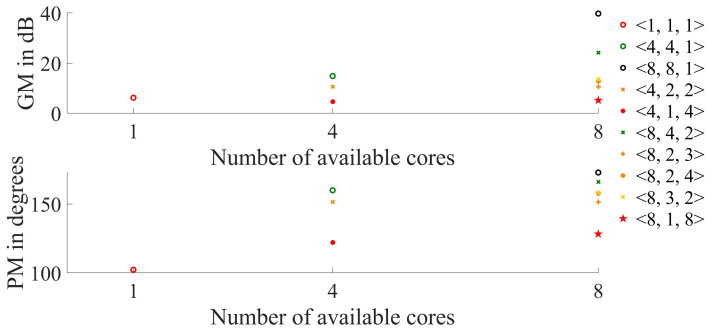



Figure 4.18: Pareto-plot for GM and PM vs $n_c^{avl}$. The legend denotes $< n_c^{avl},\ n_c^{//}, p >$.



## 4.8 Conclusions

We have presented the SPADE flow for IBC system implementation that considers both pipelining and parallelism in an integral fashion to improve QoC of multiprocessor IBC implementations. We propose model transformations for modelling, analyzing and mapping the IBC system. We explain how the SPADE approach can explicitly take into account inter-frame dependencies in pipelining, image workload variations, application parallelism, resource sharing, camera frame rate and a given platform allocation. We validate the SPADE approach using Matlab simulations considering the predictable COMPSOC platform and using hardware-in-the-loop experiments with an NVIDIA AGX Xavier platform. We observe that considering pipelined parallelism has inherent advantages over considering pipelining and parallelism separately. Exploiting parallelism should be prioritized over pipelining. But pipelined parallelism is better when an application has a limited degree of parallelism or when the inter-frame dependencies are significant. Future work may involve exploring the optimal degree of pipelining and parallelism when multiple IBC systems are sharing a platform.

# 5

# IBC design considering workload variations

5

We consider the problem of designing an image-based control (IBC) application mapped to a multiprocessor platform. Sensing in IBC consists of compute-intensive image-processing algorithms whose execution times are dependent on image workload. The challenge is that the IBC systems have a high (worst-case) workload with significant workload variations. Designing controllers for such IBC systems typically consider the worst-case workload that results in a long sensing delay with suboptimal quality-of-control (QoC). The challenge is: how to improve the QoC of IBC for a given multiprocessor platform allocation while exploiting workload variations?

In the previous chapters, we addressed this problem by considering a switched system where the switching is assumed to be arbitrary (Chapters 2 and 4) or using a compute-intensive adaptive model-predictive control (MPC) formulation based on an input-output model (Chapter 3). In this chapter, we present a controller synthesis method based on a Markovian jump linear system (MJLS) formulation considering workload variations. Our method assumes that system knowledge is available for modelling the workload variations as a Markov chain. In real-life situations, switching is not arbitrary and the switching probabilities can be modelled as transition probabilities in the Markov chain. We compare the MJLS-based method with two relevant control paradigms - linear quadratic regulator (LQR) control considering worst-case workload, and switched linear control - with respect to QoC and available system knowledge. Our results show that taking into account knowledge about switching probabilities in workload variations in controller design benefits QoC compared to the controller design methods using switched LQR presented in Chapters 2 and 4. We then provide design guidelines on the control paradigm to choose for an IBC application given the requirements and the system knowledge.

---

The content of this chapter is an adaptation of the following paper:
Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *58th Conference on Decision and Control (CDC)*, pages 3997–4004. IEEE, 2019.

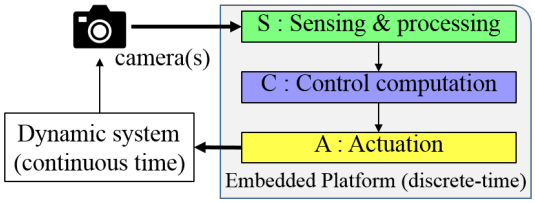



Figure 5.1: An IBC system: block diagram (repeating Fig. 1.4 (a), for readability)

## 5.1 Background and contributions

IBC systems are a class of data-intensive feedback control systems having camera(s) as the sensor (see Fig. 5.1). IBC has become popular with the advent of efficient image-processing systems and low-cost complementary metal–oxide semiconductor (CMOS) cameras with high resolution [29] [135]. The combination of the camera and image processing (sensing) gives necessary information on parameters such as relative position, geometry, relative distance, depth perception and tracking of the object-of-interest. This enables the effective use of low-cost camera sensors to enable new functionality or replace expensive sensors in cost-sensitive industries like automotive [29] [101] [110].

A typical implementation of an IBC system uses LQR control [37] and considers the worst-case workload [110]. However, this leads to a long sensing delay, poor effective resource utilisation in the multiprocessor platform, and suboptimal QoC [110]. Fig. 5.2 illustrates these challenges. The camera captures an image stream at a fixed frame rate (frames per second (FPS)). The execution times of the compute-intensive processing (sensing) of the image stream depend on image workload variations. The workload variations occur due to image content and result in a wide range between best-case and the worst-case image-processing times.

The workload variations can, however, be statistically analysed, e.g., as a PERT distribution [1] or discrete-time Markov chain (DTMC) [139], from observed data and can be modelled as workload scenarios (explained in Chapter 2). The workload scenarios can be modelled (e.g. as a task graph or a dataflow graph), analysed (for timing) and then mapped to a multiprocessor platform. A system scenario abstracts multiple workload scenarios having the same sampling period as determined by platform constraints. An optimal mapping and controller may then be designed for each system scenario.

For efficiently designing IBC systems, we should consider the workload variations and the given platform allocation. An ideal design approach should: (i) identify, model and characterise the workload scenarios; (ii) find optimal mappings for these workload scenarios for the given platform allocation; (iii) identify optimal system scenarios; and (iv) design a controller with high overall QoC for the chosen system scenarios. One of the critical aspects here is: what is a good metric to define the QoC for the application? A vision-guided braking application requires a

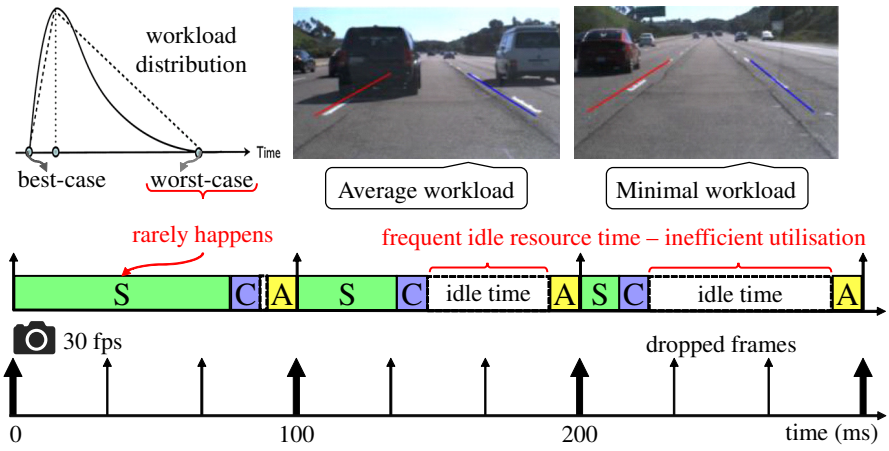



Figure 5.2: Illustration of IBC system implementation and challenges for LQR control design considering worst-case workload. S: sensing and image processing, C: control computation and A: actuation, see Fig. 5.1. (Adapted from Fig. 2.2, for readability.)



fast settling time, whereas an automotive vision-based lateral control [127] application requires to minimise the reference tracking error.

Section 1.6 introduces the scenario- and platform-aware design (SPADE) approach for designing IBC systems. SPADE characterises a set of frequently occurring workload scenarios, identifies a set of system scenarios that abstract multiple workload scenarios based on platform constraints, and designs a switched linear control system for these system scenarios to improve QoC. However, if the number of switching subsystems is high, a challenge in SPADE is the difficulty to guarantee stability for the resulting switched system [93, 135]. In case of failure to guarantee stability, SPADE as developed so far in the earlier chapters would result in LQR control for the worst-case workload scenario.

The contributions of this chapter are as follows:

- We present an alternate controller synthesis method based on a MJLS formulation for the control design step in the full SPADE approach of Chapter 4. Our synthesis method involves the following steps. (i) Modelling workload variations as a DTMC, (ii) system scenario identification, and (iii) controller design and implementation. The motivation to choose the MJLS approach [30] over other standard sampled-data linear control design techniques [100] is that it does not require us to know the exact sequence of incoming sample times due to the workload variations apriori.

- We provide design guidelines on the applicability of control design methods for given requirements, implementation constraints and system knowledge. For this, we compare the three control paradigms – optimal control

design using LQR, switched linear control design [135], and controller synthesis using the MJLS formulation – for IBC system design with respect to QoC while taking into account available system knowledge and implementation constraints, i.e., camera fps, platform allocation and mapping. Note that we cannot compare with adaptive [52] or model predictive control [15] approaches since we do not know the exact sequence of occurrence of incoming sample times due to the workload variations apriori.

## 5.2 Embedded image-based control

We consider a setting for an IBC system as shown in Fig. 5.1. Our sensor is the camera module that captures the image stream. The image stream is then fed to an embedded platform, e.g., a multiprocessor system-on-chip (MPSoC), at a fixed frame rate (fps), e.g., 30 fps. The tasks in our IBC application - compute-intensive image sensing and processing (S), control computation (C) and actuation (A) - are then mapped to run on this MPSoC.



### 5.2.1 Linear time-invariant (LTI) systems

We consider an LTI system of the following form:

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

where $x(t) \in \mathbb{R}^n$ represents the *state*, $y(t) \in \mathbb{R}$ represents the *output* and $u(t) \in \mathbb{R}$ represents the control *input* of the system at any time $t \in \mathbb{R}_{\geq 0}$. $A_c$, $B_c$ and $C_c$ represent the state, input and output matrices of the system, respectively.

We illustrate our work using the motivating case study of a vision-based lateral control system model explained in Section 1.7. We consider the LTI model with $v_x = 15 m/s$, and

$$A_c = \begin{bmatrix} -10.06 & -12.99 & 0 & 0 & 0 \\ 1.096 & -11.27 & 0 & 0 & 0 \\ -1.000 & -15.00 & 0 & 15 & 0 \\ 0 & -1.000 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, B_c = \begin{bmatrix} 75.47 \\ 50.14 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The fifth system state captures the curvature of the road at the look-ahead distance. The fifth state is required for observability of road curvature. The control input $u(t)$ is the front wheel steering angle $\delta_f$ and the output $y(t)$ is the look-ahead distance $y_L$ leading to $C_c = [0\ 0\ 1\ 0\ 0]$.

### 5.2.2 Implementation, control law and control configurations

The discrete-time control implementation we consider is already explained in Section 2.2.2. The control input $u[k]$ is a *state feedback* controller of the following form,

$$u[k] = K_{s_i} z[k] + F_{s_i} r_{ref}$$

where $K_{s_i}$ is the state feedback gain and $F_{s_i}$ is the feedforward gain both designed for the workload scenario $s_i$. $r_{ref}$ is the reference value for the controller. The approaches we use for designing the gains are explained in Section 5.5.

For each workload scenario $s_i$, we then define a control configuration $\chi^c_{s_i}$ as a tuple $\chi^c_{s_i} = (h_{s_i}, \tau_{s_i}, K_{s_i}, F_{s_i})$.

## 5.3 IBC model, mapping and configurations



In this chapter, we consider the scenario-aware dataflow (SADF) model of Fig. 2.3(a) with the same execution time parameters as our application model. The modelling, mapping and analysis of this SADF model has already been explained in detail in Chapter 2 and is not repeated here. In Chapter 2, the workload variations are characterised using a PERT distribution [1] (see distribution in Fig. 5.2). Using this information, the probability of frequently occurring workload scenarios are characterised. However, information regarding scenario transitions is not captured. This means that any arbitrary order for scenario switching sequences needs to be considered in the language of the SADF model.

In the MJLS-based approach, the workload variations are characterised using a DTMC that takes into account the scenario transition probabilities. The states of the DTMC model the workload scenarios (see Section 5.5.1) and the transitions in the DTMC model the scenario transitions. This means that the DTMC determines the language of the SADF model [129].

## 5.4 Control problem and QoC metrics

We consider a regulation problem for the IBC system. That is, the control objective is to design $u[k]$ such that $y[k] \rightarrow r_{ref}$ (a constant reference) as $k \rightarrow \infty$. The control objectives can be performance-oriented, control effort/energy-oriented or a combination of both. The control performance quantifies, in essence, how fast the output $y[k]$ reaches the reference $r_{ref}$. The control effort is the amount of energy or power necessary for the controller to perform regulation. The control performance and effort are parameters that can be tuned in the cost function for the LQR design and MJLS synthesis using the state and input weights. We evaluate QoC of an IBC application considering the following metrics: two commonly

used control performance metrics - mean square error (MSE) (explained in Section 2.2.5) and settling time (ST) (explained in Section 4.2.2) and two commonly used metrics to evaluate control effort/energy - power spectral density (PSD) and maximum control effort (MCE).

**Power spectral density (PSD):** The PSD of a signal describes the power present in the signal as a function of frequency, per unit frequency. It tells us where the average power is distributed as a function of frequency. We use Welch's overlapped segment averaging spectral estimation method [138] to compute PSD of our control input. Lower PSD for the control input signal implies that the energy required is less and hence QoC is better.

**Maximum control effort (MCE):** We define the maximum control effort as $\max_k \|u[k]\|$. A lower MCE means better QoC. MCE can also be used to identify input saturation, if any.

## 5.5 Control design



The control design technique we choose decides the controller feedback and feedforward gains $K$ and $F$ for the control law defined in Section 5.2.2. To design a controller, we assume that the sampling period $h_i$ and sensor-to-actuator delay $\tau_i$ are known.

### 5.5.1 MJLS synthesis

In this section, we characterise the workload variations as a DTMC. The states of a DTMC model the workload scenarios and the transitions model the scenario switching. We aggregate workload scenarios into system scenarios as explained in Section 2.5.3, and then recompute transition and steady-state probabilities for the DTMC. This results in a DTMC with number of states equal to the number of identified system scenarios, with each state representing a system scenario. We assume that the switching between the different control configurations is governed by this DTMC and show how the system in (5.1) can be re-cast as a MJLS [30]. For the sake of simplicity, we illustrate the formulation using only three states in the DTMC representing three system scenarios. Note, however, that it is applicable to any number of identified system scenarios (as explained in Section 2.5.3).

A Markov chain consists of a tuple $\mathscr{M} = (X, P)$ where $X$ represents the state space, and $P$ represents the one-step transition probability matrix. In our context of system scenarios annotated with sampling period and sensor-to-actuator delay, the state-space of $\mathscr{M}$ is given by $X = \{s_1, s_2, s_3\}$, where $s_i = (h_i, \tau_i), i \in \{1,2,3\}$ represent three system scenarios, and

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}.$$

Associated with $\mathscr{M}$ is a discrete-time stochastic process $\theta : \mathbb{N} \to X$ such that for all times sampling instances $k \in \mathbb{N}$, and states $s_i, s_j \in X$, $i, j \in \{1,2,3\}$, one has:

$$\text{Prob}(\theta[k+1] = s_j \mid \theta[k] = s_i) = p_{ij},$$

i.e., $p_{ij}$ represents the probability of transitioning from state $s_i$ to $s_j$. We assume that the initial condition of the stochastic process, i.e., $\theta[0]$, is deterministic.

Using a zero order sample-and-hold approach, it can be readily shown that Eq. (5.1) can be re-formulated into an MJLS governed by the following stochastic difference equations:

$$\begin{aligned} x[k+1] &= A_{\theta[k]}x[k] + B_{0,\theta[k]}u[k] + B_{1,\theta[k]}u[k-1], \\ y[k] &= C_c x[k] \end{aligned} \tag{5.2}$$

where for each $s_i \in X$, $i \in \{1,2,3\}$: $A_{s_i}$, $B_{0,s_i}$, and $B_{1,s_i}$ are computed using Eq. 2.3. In Eq. 5.2, we assume that $u[-1] = 0$ for $k = 0$. We define the new system states $z[k] = \begin{bmatrix} x[k] & u[k-1] \end{bmatrix}^T$ with $z[0] = \begin{bmatrix} x[0] & 0 \end{bmatrix}^T$ to obtain a higher-order augmented system as follows:

$$\begin{aligned} z[k+1] &= A_{aug,\theta[k]}z[k] + B_{aug,\theta[k]}u[k], \\ y_z[k] &= C_{aug}z[k] \end{aligned}$$

where for each $s_i \in X, i \in \{1,2,3\}$, $A_{aug,s_i}$ and $B_{aug,s_i}$ are computed using Eq. 2.5 and $C_{aug} = \begin{bmatrix} C_c & 0 \end{bmatrix}$.

**Infinite-horizon quadratic optimal controller:** Here, we present the control law design for the MJLS of Eq. 5.2. We design a controller to minimize the infinite-horizon cost given by

$$J(\theta[0], z[0], u[0]) = \sum_{k=0}^{\infty} \mathrm{E}[z[k]^T C_{aug}^T C_{aug} z[k] + d_u^2 |u[k]|^2],$$

where $d_u \in \mathbb{R}_{>0}$ represents the input weight and the notation $\mathrm{E}[X]$ represents the expected value of a random variable $X$.

It is shown in [30] that the solution to the above infinite-horizon optimal-control problem can be obtained by solving the *coupled algebraic Riccati (matrix) equations* (CARE)

$$\begin{aligned} &\Gamma_i = A_{aug,s_i}^T \mathscr{E}_i(\Gamma) A_{aug,s_i} + C_c^T C_c \\ &- A_{aug,s_i}^T \mathscr{E}_i(\Gamma) B_{aug,s_i} (B_{aug,s_i}^T \mathscr{E}_i B_{aug,s_i})^{-1} B_{aug,s_i}^T \mathscr{E}_i(\Gamma) A_{aug,s_i} \end{aligned}$$

where

$$\mathscr{E}_i(\Gamma) = \sum_{j=0}^{3} p_{ij}\Gamma_j$$

where $i \in \{1,2,3\}$ and $\Gamma = \{\Gamma_1, \Gamma_2, \Gamma_3\}$ are the unknown matrices to be solved for. The mean-square stabilizing optimal control law is then given by

$$u[k] = K_{\theta[k]}(\Gamma)x[k] + F_{\theta[k]}r_{ref},$$

where

$$K_{s_i} = -(B_{aug,s_i}^T \mathscr{E}_i(\Gamma) B_{aug,s_i})^{-1} B_{aug,s_i}^T \mathscr{E}_i(\Gamma) A_{aug,s_i},$$
$$F_{s_i} = \frac{1}{C_{aug}(I - A_{aug,s_i} - K_{s_i} B_{aug,s_i}))^{-1} B_{aug,s_i}},$$

where $i \in \{1,2,3\}$. It is shown in Theorem A.12 in [30] that the above CARE can be solved by solving a related convex optimization problem.

In the context of SPADE, the control configuration for a workload scenario $s_i$ is then $(\tau_{s_i},\ h_{s_i},\ K_{s_i},\ F_{s_i})$.

5

### 5.5.2 LQR design with worst-case workload

We consider the system scenario for the worst-case workload $s_{wc}$ having the worst-case period and delay $(h_{wc}, \tau_{wc})$ (one of the scenarios as explained in Section 2.5.3) for designing the control law. The worst-case system scenario for the MJLS formulation is same as $s_{wc}$. We design a controller to minimize the following cost function

$$J(u) = \sum_{k=0}^{\infty} z[k]^T d_s C_{aug}^T C_{aug} z[k] + d_u^2 |u[k]|^2,$$

where $d_u, d_s \in \mathbb{R}_{>0}$ represents the input weight and the state weights respectively. The weights are optimized for the considered QoC metric. Typically, $d_u \ll d_s$ so as to optimise for control performance and $d_u \gg d_s$ to optimise for control energy (see Section 5.6.2).

### 5.5.3 Switched linear control design

The switched linear control design we consider is explained in Section 2.2. This is the basic principle used in the controller designs presented in Chapters 2 and 4. Frequently occurring workload scenarios are characterised using the the PERT distribution. A set of optimal system scenarios are identified. LQR controllers are designed for each of these scenarios $s_s$ with $(h_s, \tau_s)$ that minimises the cost function given in Section 5.5.2. Further, the stability of this switched system is analysed by deriving linear matrix inequalities (LMIs) that check for the existence of a common quadratic Lyapunov function (CQLF).

## 5.6 Results, observations and guidelines

We consider the case-study of vision-based lateral control for a vehicle (explained in Section 5.2) for comparison of the three approaches for control design with respect to the QoC metrics described earlier. The controllers for LQR and switched linear control are tuned for the corresponding QoC metric evaluation by adjusting the input and state weights.

### 5.6.1 Simulation results

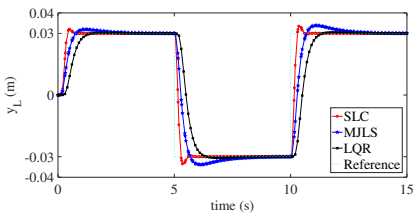
We illustrate an instance of our simulation that compares the three control paradigms: (i) for a reference output profile (for the control state $y_L$) shown in Fig. 5.3 (a), we see that the switched linear control system (SLC) settles faster than both the MJLS and LQR designs; and (ii) for control input shown in Fig. 5.3 (b), we see that minimum control effort is needed for LQR. SLC needs the highest control effort and might violate the input saturation requirements, if any. The control metrics PSD and MCE are derived from the control input $u[k]$ plots (e.g., see Fig. 5.3 (b)) and focus on minimizing the control effort or energy, whereas the control metrics MSE and ST are derived from the considered control output $y_L$ plots (e.g., see Fig. 5.3 (a)) and focus on improving the performance of the system output.

We consider for the above simulation instance a frame rate of 30 fps, i.e., $f_h = \frac{1}{30} = 33.33\ ms$ and an allocation of two processors. Then, we characterise the workload variations of a synthetic data set using a DTMC model. We notice that the $(h_i, \tau_i)$ for the worst-case workload $s_{wc}$ for this allocation is $(100, 74)\ ms$. We then identify (as explained in Section 2.5.3) the three possible system scenarios $\{s_1 = (f_h, 33),\ s_2 = (2f_h, 57),\ s_3 = s_{wc} = (3f_h, 74)\}\ ms$. The transition probability matrix of the DTMC model considered in this case for the three scenarios is:

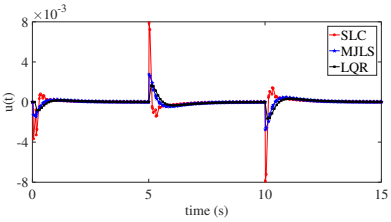
$$P = \begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.25 & 0.5 & 0.25 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}.$$

The three controllers are then designed for the above scenarios as explained in Section 5.5 using the input weight $d_u = 1$.

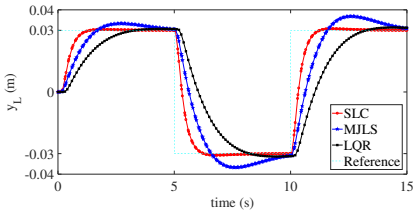
The control performance is then evaluated for the QoC metrics (defined in Section 5.4) - MSE, ST, PSD, MCE and combinations of MSE/ST with PSD/MCE. The combinations of MSE/ST with PSD/MCE is considered as they are contradictory in nature and optimising both together is a challenge and often times necessary. The above QoC metrics' empirical cost for each control technique is evaluated over multiple runs of the simulation. Each simulation run generates different workload scenario sequences that satisfy the modelled DTMC. These scenario sequences determine the switching sequence for both the SLC and MJLS control designs.



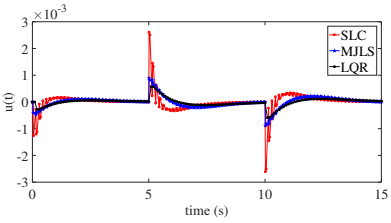

(a) Output $y_L$ when input weight $d_u = 1$.




(b) Input $u[k]$ when input weight $d_u = 1$.




(c) Output $y_L$ when input weight $d_u = 10$.




(d) Input $u[k]$ when input weight $d_u = 10$.

Figure 5.3: Comparison between controller synthesis method based on MJLS formulation (Section 5.5.1), LQR design (Section 5.5.2), and SLC system design (Section 5.5.3).

### 5.6.2 Exploration and observations

In order to provide design guidelines, we consider and vary the following different parameters: number of scenarios - we consider 3, 4 and 6 system scenarios; camera frame rates - 30 and 60 fps; input and state weights used for tuning the controllers; given platform allocation - 1, 2, 3, 4, 5, and 6 processors; and available system knowledge. We then evaluate the empirical cost of the QoC metrics for each control technique over multiple runs of the simulation. The effects of varying these parameters are explained as follows:

1. The number of states in the DTMC model, the corresponding transition probability matrix and the number of control configurations change proportionally with the number of scenarios we consider. Only the SLC and MJLS system are affected by this parameter as the LQR approach always (and only) considers the worst-case workload scenario;

2. Changing the camera frame rate affects the total number of feasible scenarios we can have. As the maximum number of scenarios we can have, intuitively, is equal to $\lceil \frac{\tau_{wc}}{f_h} \rceil$, where $\tau_{wc}$ is the sensor-to-actuator delay for the worst-case workload (for a given platform allocation) and $f_h = \frac{1}{\text{camera fps}}$;

3. The input and state weights directly affect the control performance. E.g. for the simulation instance considered before, if we set the input weight $d_u = 10$, we obtain the plots shown in Fig. 5.3 (c) and Fig. 5.3 (d). What we observe is a similar overall trend for the considered control paradigms. However, we

Table 5.1: Guidelines for choosing the control design techniques: MJLS (Section 5.5.1), LQR (Section 5.5.2), SLC (Section 5.5.3).

| Available system knowledge | QoC metrics | | | | |
|---|---|---|---|---|---|
| | Performance | | Control energy | | Performance and Energy |
| | MSE | ST | MCE | PSD | |
| Only worst-case workload information | LQR | LQR | LQR | LQR | LQR |
| Frequently occurring workloads as a PERT | SLC | SLC | LQR | LQR | SLC/ LQR |
| Frequently occurring workloads and their transition probabilities as a DTMC | SLC/ MJLS | SLC/ MJLS | MJLS/ LQR | MJLS/ LQR | MJLS |

see poorer QoC metrics for MSE and ST and better QoC metrics for PSD and MCE, when compared to $d_u = 1$ and all other parameters remaining the same;

4. The given platform allocation directly affects the timing parameters for a scenario $s_i$, i.e., $(h_i, \tau_i)$. A higher number of available processors mean that we could execute more tasks in parallel and reduce the $(h_i, \tau_i)$ (even) for the worst-case workload scenario. This means that we could possibly reduce the total number of scenarios as well;

5. System knowledge is an important parameter that determines which control design techniques can be used. An optimal control design using LQR only requires the worst-case (workload) timing information. However, designing an SLC system requires information regarding frequently occurring workloads as well and for the MJLS synthesis approach, we need both the frequently occurring workloads and their transition probabilities.

### 5.6.3 Design guidelines

The design guidelines we have inferred from observing our simulations for choosing the control design techniques for different QoC metrics and available system knowledge are listed in Table. 5.1. The cases we see in the table are explained below.

- *Only worst-case workload information is known:* This situation is quite common for a control designer. Worst-case response time or delay of the algorithms can be analysed (where often times are pessimistic) through profiling and/or analytical methods [110]. The control designer is then given only the worst-case timing information and is asked to design a controller with a QoC requirement. In this case, the SLC and MJLS approach are not applicable and only the optimal LQR design approach can be used.

- *PERT distribution is known:* Here, like we did in Chapters 2 and 4, we assume that the information with respect to frequently occurring workloads is known and are characterised analytically as a PERT distribution [1]. In this case, SLC wins for performance-oriented metrics - MSE and ST, and

LQR wins for control effort or energy-oriented metrics - PSD and MCE. For jointly optimising performance and energy, there is no clear winner as it depends mainly on which of the two metrics is more important. If performance is important, SLC is preferred and if energy is important, LQR should be chosen. The MJLS approach is not applicable as more information is needed.

- DTMC *model is known:* Information regarding frequently occurring workloads and their transition probabilities are needed for modelling a DTMC. These can be estimated from observed workload-variations data [139]. Intuitively, this means that we can predict the possible (workload) scenario switching sequences for the control design. However, for the above two cases the switching sequence is assumed to be arbitrary and not known. In this case, for performance metrics, MJLS wins when the input weight $d_u$ is very small (since SLC tends to oscillate before settling). However, for a large value of $d_u$, there is no clear winner between SLC and MJLS and it depends on the application and chosen parameters. Please note, however, that a challenge of SLC is to prove the stability of the designed system. MJLS is a synthesis method and the design, if any, is stable by construction.
  If we consider control effort or energy metrics, LQR wins when the input weight $d_u$ is small and there is no clear winner between LQR and MJLS for a large input weight $d_u$ as the results are similar and depend on the application and chosen parameters. MJLS is the clear winner if we consider a joint optimisation for performance and energy QoC metrics.



## 5.7 Conclusions

We presented a MJLS formulation for controller synthesis for image-based control systems considering workload variations and platform implementation constraints. Within the scope of SPADE, we further compare this method with two relevant control paradigms: LQR and switched linear control system design. We also provide design guidelines on the control technique to use for given constraints on the system knowledge, the QoC, and the implementation.

The synthesis method assumes that the workload variations can be characterised as a DTMC. A DTMC is sensitive to the data used for its modelling. As a future work, sensitivity analysis of the DTMC towards the QoC needs to be evaluated. Further, the current design guidelines are provided based on multiple empirical simulation runs of the controller for varying workloads, number of scenarios, camera frame rate and given platform allocation. A formal mathematical analysis would strengthen our design guidelines. The challenge for a formal analysis of the control design is that we do not know the exact sequence of occurrence of the workload variations apriori.

# 6

# Approximation-Aware Design of IBC Systems

Image-based control (IBC) systems are common in many modern applications. In such systems, image-based sensing imposes massive compute workload, making it challenging to implement them on embedded platforms. Approximate image processing is a way to handle this challenge. In essence, approximation reduces the workload at the cost of additional sensor noise. In this work, we propose an approximation-aware design approach for optimizing the memory and performance of an IBC system, making it suitable for embedded implementation. We perform compute- and data-centric approximations and evaluate their impact on the memory utilization and closed-loop quality-of-control (QoC) of the IBC system. Further, an IBC system operates under several environmental scenarios, e.g., weather conditions. We evaluate the sensitivity of the IBC system to our approximation-aware design approach when operated under different scenarios and perform a failure probability (FP) analysis using Monte-Carlo simulations to analyze the robustness of the approximate system. Further, we design an optimal approximation-aware controller that models the approximation error as sensor noise and show QoC improvements. In essence, this is an alternative design paradigm in the scenario- and platform-aware design (SPADE) flow to deal with high compute workload, complementary to parallelisation and pipelining. We demonstrate the effectiveness of our approach through the lane-keeping assist system (LKAS) case study using a heterogeneous NVIDIA AGX Xavier embedded platform in a hardware-in-the-loop (HIL) framework. Both the platform and the LKAS case study were already introduced in Chapter 1. We show substantial



The content of this chapter is an adaptation of the following three papers:

1. Sayandip De, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 8: 174568–174586, 2020.
2. Sayandip De, Sajid Mohamed, Konstantinos Bimpisidis, Dip Goswami, Twan Basten, and Henk Corporaal. Approximation trade offs in an image-based control system. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1680–1685. IEEE, 2020.
3. Sajid Mohamed, Sayandip De, Konstantinos Bimpisidis, Vishak Nathan, Dip Goswami, Henk Corporaal, and Twan Basten. IMACS: A Framework for Performance Evaluation of Image Approximation in a Closed-loop System. In *8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2019.

memory reductions and QoC improvements with respect to the accurate implementation. Approximate computing also helps to improve energy efficiency of our design. This last aspect of the work, reported in [36], is excluded in this thesis as it is not the primary focus of this thesis.

## 6.1 Background and contributions

IBC systems are feedback control systems that rely on image-based sensing data obtained from (a) camera sensor(s). Advancements in camera technologies, image-processing algorithms and parallel computing heterogeneous platforms have made IBC systems immensely popular in automotive applications [16] like advanced driver assistance systems (ADASs), autonomous driving systems, etc. However, the enormous compute requirements of IBC systems make them challenging to be implemented on such platforms without sacrificing control performance. The focus of this work is to improve the compute and memory efficiency of IBC systems (implemented on embedded platforms) by introducing approximations, without sacrificing the control performance. In essence, approximation reduces the compute workload at the cost of additional sensor noise. However, the inherent error resilience of IBC systems allows the introduction of this additional sensor noise without sacrificing control performance.



A typical IBC system consists of a sensing task (S), a control task (C) and an actuation task (A) (see Fig. 6.1). S involves pre-processing the image frames obtained from the camera sensor and extracting application-specific features. C applies the control algorithm using these extracted features and A implements the control decisions in the environment. A worst-case execution time (WCET) analysis shows that the execution time of S is orders of magnitude higher than those of C and A, thus, resulting in a long sensing-to-actuation delay $\tau$ (see Fig. 6.1).

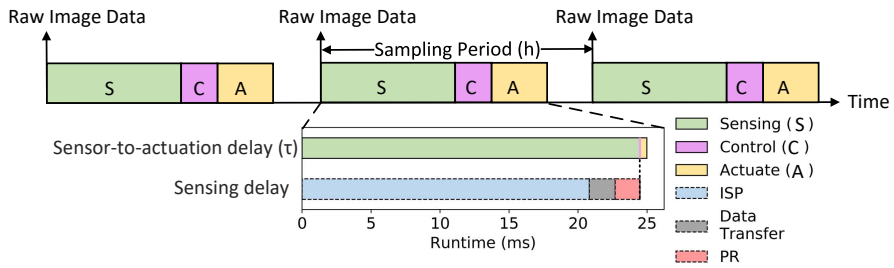



Figure 6.1: Tasks in an IBC system: Runtimes for the sub-tasks are shown for a LKAS implemented on NVIDIA AGX Xavier embedded platform [47] (8-core central processing unit (CPU)+graphical processing unit (GPU)). Runtimes are shown for 512 × 256 resolution images.

The QoC of an IBC system depends on the sensing-to-actuation delay $\tau$.

Smaller $\tau$ results in a lower sampling period for the controller, which improves the overall QoC. The sensing task (S) is the main bottleneck in lowering $\tau$. Besides through parallelisation, as already explained in Chapter 2, approximate computing can reduce the WCET of S by reducing the compute while introducing sensor noise. IBC systems are equipped with image-signal processing (ISP) pipelines optimized for human visual consumption; control algorithms, which are inherently resilient to sensor noise, do not need the high-quality images produced by these pipelines. *The work in this chapter focuses on system approximation by reducing the compute workload of the* ISP *and the data transfer traffic to off-chip memory, resulting in better* QoC *of the overall system.*

Reduced compute workloads due to approximations introduce new task mapping opportunities for parallelised sensing tasks. Approximate tasks can be mapped to slower power-efficient platform cores, while operating the controller at the same sampling period, guaranteeing proper system functionality. The combined impact of both approximations and platform mappings was not explored in the literature prior to this work. *The work in this chapter explores the interplay between the degree of approximation and different platform mappings while analyzing their impact on* QoC *and memory efficiency of the entire* IBC *system.*

A point to consider is that approximation introduces errors in the measurement of states of the system. *We propose a method to design an approximation-aware optimal linear-quadratic-Gaussian (*LQG*) controller by modelling the approximation error as sensor noise.*

IBC systems operate under different environmental scenarios [59]. For example, image feedback at night requires a different nature of processing compared to the same during the day in ADAS. These scenarios significantly influence the degree of approximation that can be tolerated without destabilizing the system. The robustness study on how approximations impact the IBC performance under different environmental scenarios is essential but is was also not addressed in prior literature. *For different environmental conditions, we design different approximation-aware controllers and show that our approach is robust to different environmental scenarios by performing a* FP *analysis for the approximate system using Monte-Carlo simulations.*

In summary, our approach takes into account various artefacts of approximation-in-the-loop in terms of platform mapping and controller design, while considering robustness to failures. We refer to this approach as approximation-aware IBC design. The key contributions of this work are as follows:

1. *Performance evaluation for IMAge-based Control Systems (*IMACS*) framework:* The IMACS framework[1] (Section 6.3) helps to test and evaluate the impact of approximation in closed-loop feedback systems. First, the resilience of the given IBC system to different approximation choices is anal-

[1] The IMACS framework is open sourced and can be accessed on github: https://github.com/sajid-mohamed/imacs

ysed. Second, for each approximation choice, the sensing delay is computed, and the error due to approximations is quantified using the IMACS framework.

2. *Optimized approximation-aware* IBC[2]*:* We illustrate the potential of exploiting the inherent error resilience of IBC systems by performing coarse-grained computation skipping in the ISP (compute-centric approximation, Section 6.6.2). Further, we introduce two optimizations on top. First, we perform a data-centric approximation by varying the degree of lossy compression post-ISP (Section 6.6.3), which shows a memory reduction of up to 88%, and hence a performance improvement. Second, we design an approximation-aware LQG controller that models the errors due to approximation as sensor noise (Section 6.7). These approximations and optimizations turn out to substantially improve the overall QoC compared to the accurate implementation.

3. *Scenario-awareness:* Environmental scenarios significantly influence the degree of approximation that can be tolerated without destabilizing an IBC system. We perform scenario-specific optimization considering six different environmental scenarios relevant for LKAS, i.e., *day, night, dawn, dusk, fog, snow*. We show that scenario-specific approximation decisions improve the overall QoC (Section 6.9).

4. *Failure probability analysis:* Applicability of our proposed approach to safety-critical systems (such as LKAS) require FP analysis to comply with well-accepted safety margins [17]. In this work, we perform FP analysis based on Monte-Carlo simulations to show the robustness of approximate IBC systems designed using our approach (Section 6.10).



Improving the runtime performance of the sensing task through approximations is complementary to the SPADE approach presented in earlier chapters. The SPADE approach for an industrial platform (explained in Section 4.7) already uses the approximated sensing algorithm introduced in this chapter. The approximation-aware design offers alternatives in terms of sensing (with approximate ISP) and controller (i.e., approximation-aware controller) in the SPADE flow.

## 6.2 Related work

Prior efforts in the approximate computing domain can be broadly classified into compute-centric and data-centric approaches.

**Compute-centric approximations:** Compute-centric efforts are focused on reducing the compute workload across algorithm, architecture and circuit levels.

[2]Our approximation-aware design framework is open sourced and can be accessed on github: https://github.com/sayandipde/approx_ibc

Commonly used algorithmic approximations are computation skipping [83], precision scaling [131], and replacing error-resilient compute-intensive functions with neural networks [42]. A similar learning approach to design ISPs for new camera systems is proposed in [62]. Next, at the architecture level, research efforts have focused on both approximating general-purpose processors [27] as well as domain-specific accelerators [33]. At the circuit level, research efforts focus on manual design techniques for adders and multipliers [63], as well as automated methodologies for designing energy-efficient approximate circuits [34].

**Data-centric approximations:** Data-centric approximations either approximate the memory device that is being accessed or they approximate the value of the data being accessed. Both cases lead to reduced on/off chip data traffic, thereby reducing the required memory bandwidth. Reducing the dynamic random access memory (DRAM) refresh rate [105] and load value speculation [111] are examples of approximating the memory location. Approximating data values involves storing/accessing data in a compressed format [67]. Quality-aware memory controllers for directing memory transactions to different compression schemes are proposed in [108].

Both compute and data-centric approaches are focused on approximating individual subsystems. Individual subsystems are usually a part of bigger IBC systems. Proper interaction between them is key in ensuring system stability. However, approximating a subsystem might result in undesired behaviour in another, thereby resulting in the failure of the entire IBC system. This is a major downside of approximating each subsystem as a stand-alone entity.



To address these limitations, this chapter proposes a holistic full-system analysis approach wherein different subsystems are approximated together in a compute or data-centric manner and the quality implications are evaluated for the full-system rather than individual subsystems. An overview of prior efforts in full-system approximation analysis highlighting the key differences from our work is given below.

**Full system approximation analysis:** These approaches are targeted at different application domains. For this study, we focus mainly on camera-based systems. Approximation benefits in a camera-based biometric security system, using an iris recognition application, is showcased in [57]. An approximate smart camera system is introduced in [104], using camera resolution scaling, reducing memory refresh rate and computation skipping. An approximate ISP pipeline tuned for computer-vision algorithms is designed in [21], by skipping selected ISP stages. An algorithm-hardware co-designed system is showcased in [145]. It leverages the temporal motion information generated by ISPs to reduce the compute demands of the perception engine, at the cost of accuracy loss.

These research efforts [21, 57, 104, 145] lack a closed-loop feedback behaviour. Approximation decisions in a closed-loop system have quality implications at a later point in time. Optimising a system while accounting for the temporal approximate behaviour is not explored in [21, 57, 104, 145], making this a key distinguishable feature of our work. Additionally, looking solely from an approximation per-

Table 6.1: Qualitative comparison with state-of-the-art system-level approximation approaches.

| Contributions | [57] | [104] | [21] | [145] | Our work |
|---|---|---|---|---|---|
| **compute-centric optimizations** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **data-centric optimizations** | | ✓ | | | ✓ |
| **closed-loop approximations *** | | | | | ✓ |
| evaluation framework | | | | | SIL, HIL |
| platform mappings | | | | | ✓ |
| environmental scenarios | | | | | ✓ |
| approximation-aware controller | | | | | ✓ |
| **real-system implementation** | ✓ | ✓ | ☑ | | ☑ |
| camera sensor | ✓ | ✓ | △ | △ | △ |
| computation | ✓ | ✓ | ✓ | △ | ✓ |
| controller | | | | | ✓ |
| actuation | | | | | △ |

✓ True, ☑ Partially True, △ Modeling or Simulation.
SIL: Software-in-the-loop, HIL: Hardware-in-the-loop
* *Closed-loop approximations: approximations in closed-loop feedback systems.*



spective, some of the approximation techniques applied in these research efforts can also be applied to our work for additional benefits. For instance, fine-grained computation skipping techniques proposed in [104,106] can be applied in the ISP, which is an interesting future research direction. Also, leveraging motion information to relax the number of invocations of the perception stage (as shown in [145]) that typically follows the ISP stage is another interesting research direction.

Table 6.1 summarizes the key contributions of this work stacked up against other state-of-the-art full system approximation approaches.

## 6.3 The IMACS framework

We consider a motivating case study of a LKAS to demonstrate our approximation-aware IBC design approach. The IMACS framework helps to test and evaluate the impact of approximation in closed-loop feedback systems. First, the resilience of the given IBC system to different approximation choices is analysed. Second, for each approximation choice, the sensing delay is computed, and the error due to approximations is quantified using the IMACS framework. Then, an approximation-aware controller is designed, for each approximation choice, by considering the sensing delay and modelling the quantified error due to approximations as sensor noise (explained later in Section 6.7).

Fig. 6.2 illustrates the IMACS HIL simulation setup for LKAS. The detailed description of the IMACS framework is reported in [87]. The IMACS framework supports X-in-the-loop (XIL) simulation and is not just limited to HIL simulation. It

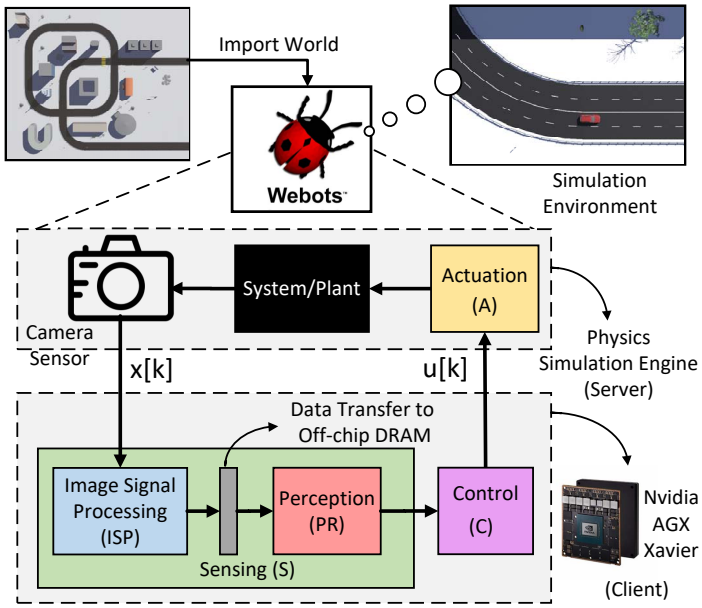


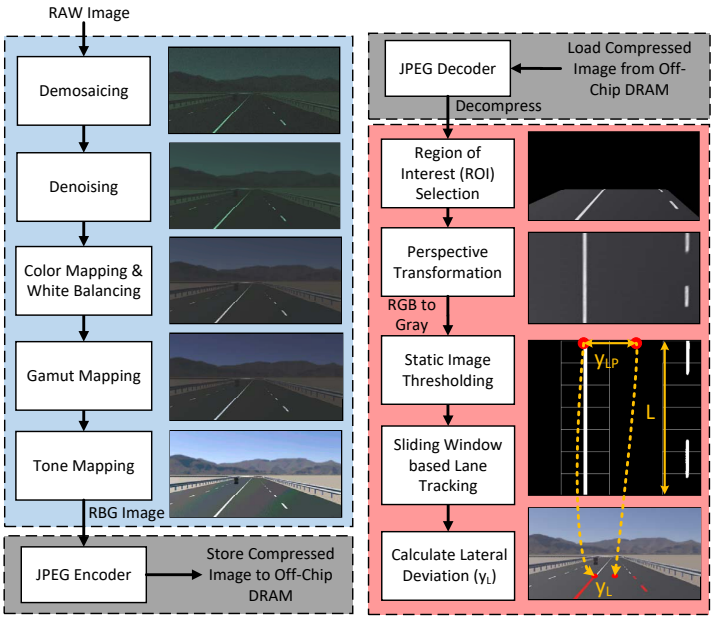

Figure 6.2: IMACS HIL simulation setup for the LKAS system.



simulates a vehicle with a top look ahead camera using a physics simulator engine, e.g.. Webots [84], and interacts with a hardware platform, e.g.. an NVIDIA AGX Xavier platform, using the transmission control protocol/internet protocol (TCP/IP) protocol. The simulator works in a server-client configuration, wherein the physics simulator engine acts as the server while the platform acts as the client. The server (Webots) progresses simulation in full synchronization with the client (NVIDIA AGX Xavier) [47]. At each simulation step, the camera sensor simulated in Webots generates a raw image containing state information $x[k]$, that is fed to the NVIDIA platform. It executes the sensing (S) and control (C) tasks to generate control input $u[k]$, which is communicated back to Webots for actuation. After actuation, the simulation progresses to the next step.

For the scope of this chapter, the IMACS framework uses the following configuration settings. The camera sensor in the Webots simulator is modelled based on the AR1335 complementary metal–oxide semiconductor (CMOS) digital image sensor [6] and is set to a resolution of 720p[3]. The camera frame rate is varied between 30 fps, 60 fps and 120 fps, depending on the sampling period of the controller. The actuation dynamics are modelled based on [107]. The vehicle is

[3]We observe that state-of-the-art lane detection algorithms [98] operate on low-resolution images. So, we perform our evaluation using downscaled (512×256) sensor images. We believe our approach is also effective for high-res images required in applications like object detection.




(a) Image Signal Processing (ISP) (b) Perception (PR)

Figure 6.3: Overview of ISP and perception (PR) stages with their corresponding outputs.

initially positioned with a fixed bias of 15 cm from the lane centre to test the control performance. A lane width of 3.25 m is considered, as per standard road safety guidelines. The Webots simulation step is set to 1 ms, while the vehicle speed $v_x$ is set to 50 km/hr for all of our evaluations.

## 6.4 LKAS implementation: overview

A LKAS consists of six main components/stages: Camera Sensor, ISP, data compression, PR, control (C) and actuate (A), as shown in Fig. 6.2. The camera sensor and actuation are modelled and executed in Webots. ISP, data compression, PR and control are executed on the NVIDIA AGX Xavier platform. Below, we provide an overview of these stages. The ISP and PR stages are also illustrated in Fig. 6.3.

### 6.4.1 Image-signal processing (ISP) and perception (PR)

An ISP pipeline transforms a RAW image in the Bayer domain to pixels in the RGB domain through a series of image enhancing stages. Modern ISPs comprise of hundreds of proprietary stages. However, in this work, we consider a set of five essential stages common to all ISP pipelines, demosaic, denoise, color map, gamut

map and tone map, as defined in [21]. Fig. 6.3(a) shows these five stages along with their corresponding outputs. It is worth noting that the RGB output from the ISP pipeline is typically stored in the main memory (off-chip DRAM) due to the large size of the image data. In this work, we consider JPEG compression (see Fig. 6.3) to reduce the data communication between different processing stages like ISP, PR, etc.

The perception (PR) stage calculates the lateral deviation of the vehicle from the centre of the lane by performing preprocessing, feature extraction and inference steps on the decompressed ISP output. During *preprocessing*, first, the region-of-interest (ROI) is selected based on the scene. A perspective transform is then performed on the ROI to get a bird's eye view of the lane ahead (see Fig. 6.3 (b) block 2 of the PR stage). During *feature extraction*, the candidate lane pixels are extracted from the bird's eye view image. For this, the bird's eye view image is converted to grayscale, and subjected to binarization using static thresholding (see Fig. 6.3 (b) block 3 of PR). Finally, candidate lane pixels are obtained using sliding windows ranging from bottom to top of the image. During *inference*, first, the previously identified lane positions markers are fit to a second-order polynomial. Then, these polynomials are used to calculate the centre of the lane at a look-ahead (L) distance. The centre of the image in the x-direction is considered as the vehicle's current position. Using these two metrics, the lateral deviation in the transformed domain ($y_{LP}$) is calculated (see Fig. 6.3 (b)). A reverse perspective transform gives the final lateral deviation ($y_L$) (see Fig. 6.3 (b))



### 6.4.2 Discrete-time control implementation (C)

We consider the bicycle model introduced in Section 1.7 for simulating the LKAS and it is described as follows,

$$\dot{x}(t) = A_c x(t) + B_c u(t),$$
$$y(t) = C_c x(t),$$

where the following vehicle parameters in Section 1.7 are adapted for the BMWX5 car model in Webots (which are parameters of system matrices $A_c$, $B_c$ and $C_c$): $l_f$, $l_r$ (= 1.6975 and 1.2975 m respectively) denote distance of the front and rear axles from the centre of gravity (CoG); $I_\psi$ (= 6337.74 kg·m$^2$) is the total inertia of the vehicle around its CoG; $c_f$, $c_r$ (= $2 \times 60000$ N/rad) denote cornering stiffness of the front and rear tires; and the total vehicle mass is $m$ (= 2000 kg).

As in earlier chapters, we guarantee constant sensing-to-actuation delay $\tau$ by enforcing an implementation with time-triggered activation of tasks. An implementation is annotated with a pair $(h_i, \tau_i)$ that models the sampling period and delay associated with it. The zero-order hold (ZOH) method is used to discretize the system [48] with the annotated $(h_i, \tau_i)$ to obtain an augmented system of the form: $z[k+1] = A_d z[k] + B_d u[k]$, where $A_d$ and $B_d$ are discretized matrices and the augmented system states $z[k] = [x[k] \;\; u[k-1]]^T$.

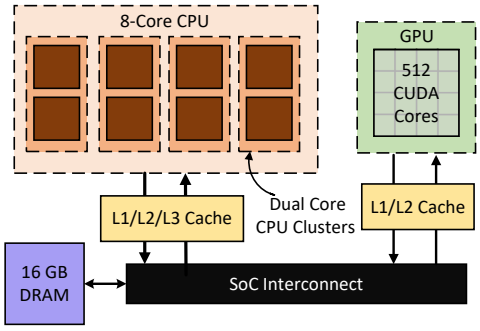
**Control law:** The control input $u[k]$ is a state feedback controller of the form

$$u[k] = Kz[k],$$

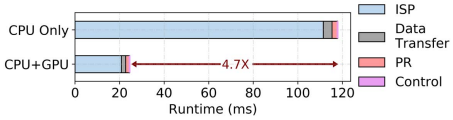
where $K$ is the state feedback gain. We design $K$ using the optimal linear quadratic regulator (LQR) [48]. The control objective is for the output $y[k] \to 0$, when $k \to \infty$.

### 6.4.3 Hardware support for LKAS

An industrial embedded heterogeneous platform NVIDIA AGX Xavier [47] is considered for LKAS implementation, as explained in Section 1.3.3 and illustrated in Fig. 6.4 (a). Note that Fig. 6.4 (a) only shows the IPs used in this work.




(a) Simplified Block diagram of NVIDIA AGX Xavier




(b) Speed up from optimized task mapping

| | CPU | GPU |
|---|---|---|
| ISP | • Input data loading | • Demosaicing<br>• Denoising<br>• Color Map<br>• Gamut Map<br>• Tone Map |
| JPEG Encode | • Formatting output codestream | • Forward DCT<br>• Huffman encoding |
| JPEG Decode | • Parsing input codestream | • Inverse DCT<br>• Huffman decoding |
| PR | • Sliding window based tracking<br>• Inference | • Preprocessing<br>• Image thresholding |
| C | • Entire control | |

(c) CPU-GPU task partitioning

Figure 6.4: LKAS task mappings on an 8-core CPU+GPU default configuration (CPU_-8C+GPU). A comparison with software only (8-core CPU) configuration is shown. Runtimes are shown for image workloads of (512x256) resolution.

**Baseline Task Mapping:** The main tasks in the LKAS that are executed on the NVIDIA platform are ISP, JPEG encode/decode, PR and control (C). As an initial step, we map all the tasks to an 8-core CPU only configuration. The measured runtimes of the individual tasks are shown in Fig. 6.4(b). The ISP takes most of the computation time. So, we map all the ISP tasks to the GPU (see Fig. 6.4 (c)). The ISP is optimized using Halide [102] domain-specific language with GPU as backend. Additionally, we also map parts of the JPEG encoding/decoding and PR to the GPU. Fig. 6.4 (c) gives a detailed mapping overview. Task offloading from CPU memory to GPU memory is a major bottleneck. We make use of the unified

memory (a single memory address space accessible from any processor in a system) support in NVIDIA Volta GPUs to optimize our mappings further. The control task C is light in compute, so we map it to the CPU. This CPU-GPU task mapping gives a runtime speedup of 4.7× over the initial 8-core CPU only mapping (see Fig. 6.4 (b)). *We consider this as a baseline for exploring approximation opportunities in* LKAS.

## 6.5 Design and evaluation strategies

In this section, we first outline our approximation-aware design strategies for optimizing IBC performance (QoC) and memory utilization. We determine what to approximate by identifying the error-resilient stages that can give maximum benefits on approximation (Section 6.5.1). Then, we explain how to approximate by summarizing the main steps of our approximation-aware IBC design (Section 6.5.2). We then proceed to show how to interpret the outcomes by introducing a QoC-optimal mode for approximated IBC systems (Section 6.5.3). Finally, we describe the quality metrics considered for evaluation (Section 6.5.4).



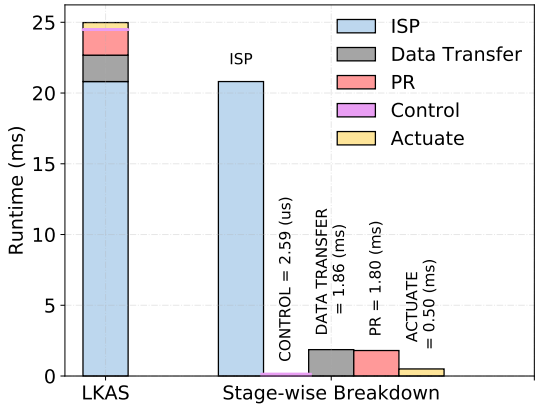



Figure 6.5: Full system runtime profiling of LKAS on NVIDIA AGX Xavier with default configuration (CPU_8C+GPU).

### 6.5.1 What should we approximate?

The first challenge is to identify the error-resilient as well as compute-heavy stages in LKAS. These are target candidates that can give maximum gains. LKAS consists of six different stages, starting with the image capture by the camera sensor and ending with the actuation, as shown in Fig. 6.2. Actuation cannot be approximated as it depends on vehicle dynamics. However, prior literature has shown that

the other stages (camera sensor [21], ISP [42], data compression [104], PR [33], control [103]) can be approximated. So, to figure out the best approximation opportunities, we perform runtime profiling of LKAS.

Runtime profiling results for LKAS are shown in Fig. 6.5. Profiling is performed using the default configuration of the NVIDIA AGX Xavier platform (see Fig. 6.4 (a)). The system is running Ubuntu 18.04. For **runtime profiling**, we execute each stage in LKAS 100 times and for 100 different images to reduce sensitivity to access locality. For each stage, we consider the maximum of all such execution runs to get the WCET. For actuation, a WCET of 0.5 ms is considered [107].

From Fig. 6.5, it is evident that ISP, which consumes 83% of the total runtime, is the main target candidate for approximation. Additionally, the off-chip data transfer can also be optimized to obtain added gains. So, in the rest of the chapter, we confine our scope to optimizing the ISP and the off-chip data transfer.

### 6.5.2 How to approximate and optimize LKAS?

In this work, we propose an approximation-aware design approach shown in Fig. 6.6. Each of the contributions presented in this chapter is marked.



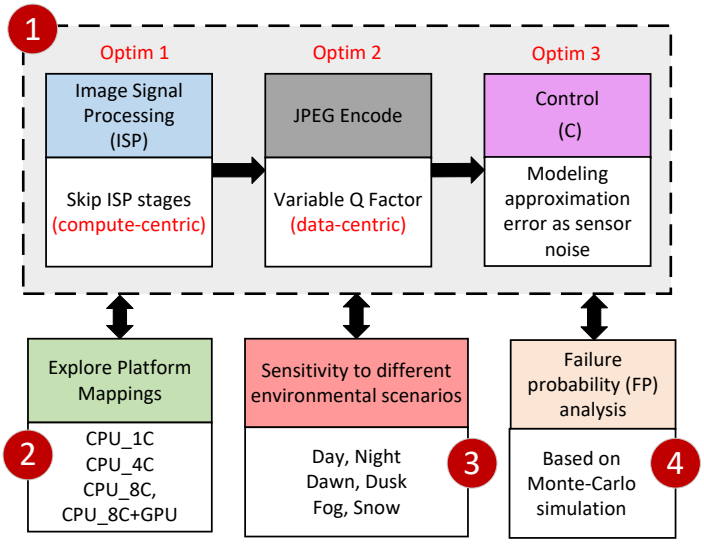



Figure 6.6: Proposed approximation-aware design approach for IBC systems.

First, we perform coarse-grained approximations to the ISP (Optim 1) by skipping one or more sub-stages within the pipeline (see Fig. 6.3 (a)). This is a compute-centric approximation approach focused on reducing the compute

workload of the ISP [4]. Then we perform data-centric approximations by varying the degree of lossy JPEG compression (Optim 2). The goal is to reduce the data transfer traffic between the processor and off-chip DRAM, as accessing DRAM is slow. The Q-parameter in the JPEG algorithm decides the degree of lossy compression. A smaller value of Q denotes larger numerical values in the quantization matrix, thereby leading to a higher level of compression. This comes at the cost of larger errors in the decompressed image. In case of Optim 1, we choose the highest value of Q (=100) and keep it constant, which results in a lower degree of lossy compression. However, in the case of Optim 2, we use the Q-parameter as a quality control knob and reduce it in discrete steps of 10 from Q = 100 to Q = 10. This reduces the off-chip data transfer traffic and impacts the QoC, and memory footprint of LKAS.

From a control-design perspective, approximations performed in Optim 1 and Optim 2 introduce errors in the state of the system. The controller is unaware of this added error. So, we design an approximation-aware LQG controller (Optim 3) by modelling the error as sensor noise. Profiling results in Fig. 6.5 show that LKAS is compute-bound. So, we expect most gains from Optim 1 as it reduces the compute workload of the system. We start by evaluating Optim 1. Then we incrementally add Optim 2 and Optim 3. It is worth mentioning that Optim 1 and 2 are not novel approximation strategies. However, the focus of this work is to evaluate the combined impact of these optimizations (Optim 1, Optim 2, and Optim 3) on the closed-loop QoC and memory of the LKAS, which is not explored in prior literature.



LKAS operates under different environmental scenarios. We design approximation-aware controllers for each scenario and evaluate the sensitivity of LKAS to Optim 1, Optim 2, and Optim 3 when operated under these scenarios. We set up six different environmental scenarios (day, night, dawn, dusk, fog and snow) in Webots for analyzing the sensitivity of LKAS to approximation errors. LKAS is a safety-critical application that requires an evaluation of its robustness to approximation error. So, we perform a FP analysis of approximate LKAS using HiL based Monte-Carlo simulation.

### 6.5.3 How to analyze approximation-aware design outcomes?

To properly analyze the design points obtained from our approximation-aware design approach, we consider two different modes: *approximation-only* mode and QoC-*optimal* mode. Fig. 6.7 shows a snapshot of LKAS operating in these modes over a fixed evaluation time window ($t_{ETW}$). In *approximation-only* mode, we consider a reduced sensing-to-actuation delay $\tau$ obtained due to approximations. However, the sampling period $h$ is kept constant. This mode helps to evaluate the advantage of only the approximation on the QoC. In QoC-*optimal* mode, both re-

[4] Note that this compute-centric approach has no impact on the data transfer traffic as the degree of lossy compression performed in the JPEG encoder post-ISP is fixed to the minimum.

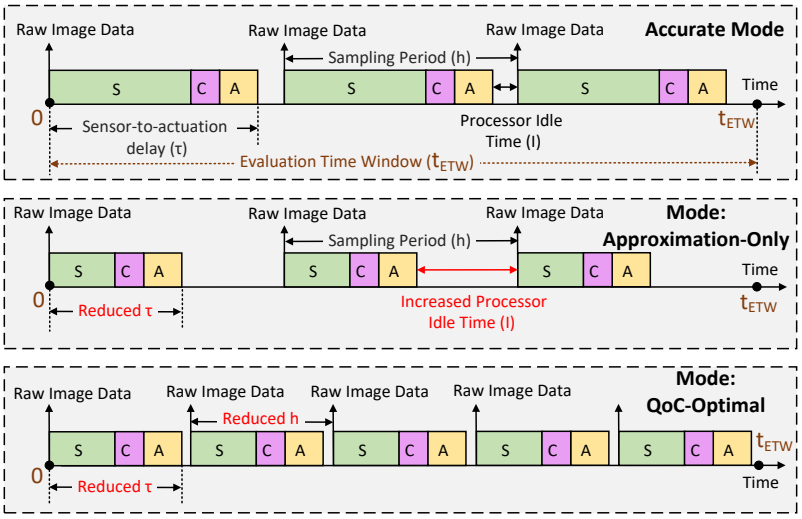



Figure 6.7: Two LKAS operational modes considered in this work: approximation-only and QoC-optimal. The fully accurate mode is shown as a reference to compare the two modes.



duced sensing-to-actuation delay $\tau$ as well as reduced sampling period $h$ are considered. This allows a higher number of frames to be processed over the same $t_{ETW}$.

### 6.5.4 Quality metrics

Image quality degradation due to Optim 1 and Optim 2 is evaluated using the *structural similarity (*SSIM*)* index. The index for two images $m$, $n$ is defined as:

$$SSIM(m,n) = \frac{(2\mu_m\mu_n + C_1)(2\sigma_{mn} + C_2)}{(\mu_m^2 + \mu_n^2 + C_1)(\sigma_m^2 + \sigma_n^2 + C_2)}$$

where $\mu_m$, $\mu_n$, $\sigma_m$, $\sigma_n$ and $\sigma_{mn}$ are the local means, standard deviations, and cross-covariance for images $m$, $n$. $C_1$, $C_2$ are constants. High structural similarity (SSIM) loss denotes images with higher visual difference. QoC evaluation of the proposed IBC system is performed using the metrics settling time (ST) (as explained in Section 4.2.2), power spectral density (PSD), maximum control effort (MCE) (as explained in Section 5.4) and mean absolute error (MAE) (as defined below). These metrics consider both control performance and control energy.
**Mean absolute error (MAE)**: mean of the cumulative sum of absolute errors, i.e.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y[k] - r_{ref}|$$

where $n$ is the no. of samples and $y[k]$ is the value of the $k^{th}$ output. A lower MAE implies a better QoC.

For all evaluations, we consider the following as defaults, unless otherwise mentioned: (a) Platform Config: CPU_8C + GPU (see Fig. 6.4 (a)) (b) Scenario: day.

## 6.6 Compute- & data-centric approximations

In this section, we evaluate the impact of approximations on the QoC and memory of LKAS. First, we explain the scheduling of approximate ISP pipelines on GPU. Then, we perform both compute-centric and data-centric approximations.

### 6.6.1 Scheduling of approximate ISP pipelines on GPU

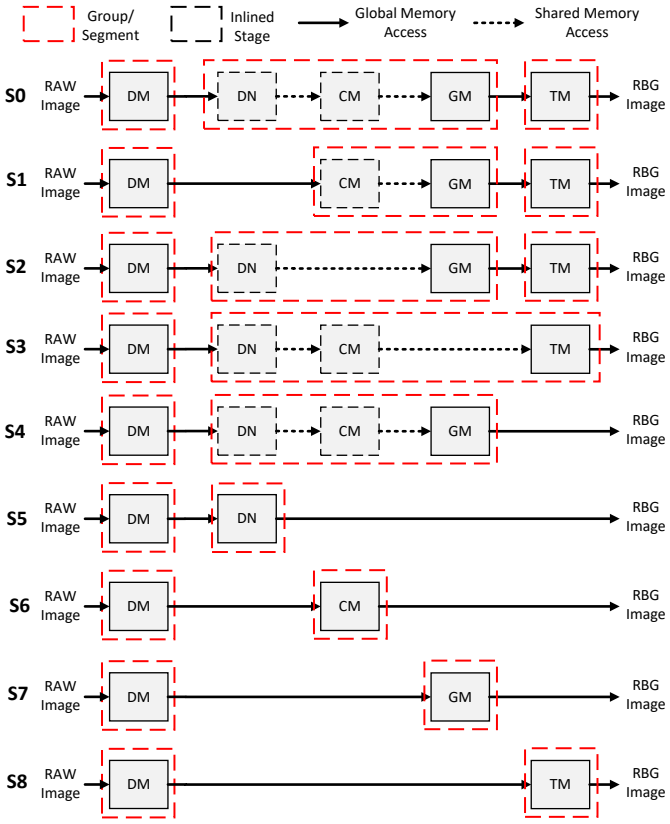



Figure 6.8: GPU schedules obtained for the different pipelines.

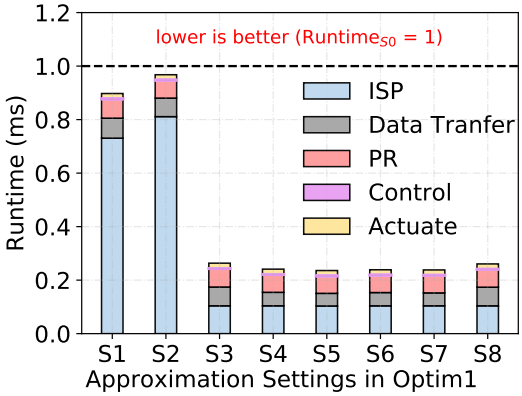



Figure 6.9: Improvements in runtime from coarse-grained ISP approximation.

Execution time of different ISP pipeline settings depends on how they are scheduled on the GPU. To generate optimized schedules, we use the Halide GPU auto-scheduler proposed in [118]. The cost function of this auto-scheduler optimizes a pipeline by splitting it into groups/segments so that each segment can be executed with accesses to the GPU shared memory only. Also, each segment corresponds to a different CUDA kernel. When moving from one segment to another, we need accesses to the global memory. It should be noted that accesses to the global memory are orders of magnitude more costly compared to the ones to shared memory since DRAM bandwidth is often much lower than the bandwidth achieved by shared memory [118]. That's why the execution time of these pipeline settings highly depends on the number of global memory accesses.

We consider nine different pipeline settings, S0-S8. S0 is the fully accurate ISP pipeline; the other eight pipelines are approximated by skipping various stages, as further elaborated in the following subsection. Fig. 6.8 shows the GPU schedules we obtain for the different pipeline settings using the auto-scheduler [118]. For splitting the pipelines into segments, the auto-scheduler starts from the output stage and continuously merges it with the previous stages as long as the data can be fit into the shared memory. If not, then it splits the pipeline into two segments and starts the same process again in the non-visited segment. The following observations are made from the schedules:

- Gamut mapping (GM) and tone mapping (TM) cannot be put into a single segment as the data cannot fit in shared memory. So, TM is put in a separate segment in S0-S2.
- The output of demosaicing (DM) has a high memory footprint. So, DM cannot be put in a single segment with the subsequent stages. This results in a separate segment for DM in S0-S8.
- The auto-scheduler prioritizes inlining of cascaded stages (within a seg-

ment) into the consumer, which maximizes producer-consumer locality.

Fig. 6.9 shows the runtimes of the eight pipeline settings. We observe from Fig. 6.8, that settings S0-S2 have three segments/groups accessing the global memory while the rest, S3-S8, has two segments/groups accessing the global memory. This explains the higher execution time of S0-S2 compared to S3-S8 in Fig. 6.9. This also explains why S3 and S4 (each with four stages) take the same time as S5-S8 (each with two stages). Also, the execution time does not scale linearly with the number of stages in the pipeline, which explains why all of S5-S8 combined seems to take less time than S0 (or S1 and S2) despite having more stages (as DM is common to all of them). It should be noted that the slight runtime variations in S0-S2 are due to extra thread synchronization overheads introduced by intra-segment communication through shared memories. The same reasoning applies to S3-S8.

Table 6.2: Coarse-Grained Approximation Settings in Optim 1.

| Setting | ISP Stages | Description |
|---|---|---|
| S0 | DM, DN, CM, GM, TM | Accurate (all stages included) |
| S1 | DM, CM, GM, TM | Skip Denoising |
| S2 | DM, DN, GM, TM | Skip Color Mapping |
| S3 | DM, DN, CM, TM | Skip Gamut Mapping |
| S4 | DM, DN, CM, GM | Skip Tone Mapping |
| S5 | DM, DN | Keep only Denoising |
| S6 | DM, CM | Keep only Color Mapping |
| S7 | DM, GM | Keep only Gamut Mapping |
| S8 | DM, TM | Keep only Tone Mapping |

DM: Demosaic, DN: Denoise, CM: Color Mapping,
GM: Gamut Mapping, TM: Tone Mapping
Note: We refer to S0 as the accurate setting as no stages are skipped. In the other (approximate) settings, S1-S8, certain stages are skipped.



### 6.6.2 Optim 1: coarse-grained ISP approximations

As mentioned, the ISP is approximated in a coarse-grained manner by skipping one or more sub-stages within the pipeline (see Fig. 6.3 (a)). Testing all possible combinations for skipping sub-stages is not feasible due to high compute overheads. So, for our analysis, we consider nine different approximation settings, as shown in Table 6.2. Settings S1-S4 are obtained by skipping one stage at a time, while settings S5-S8 are obtained by keeping one stage and disabling the rest of the pipeline. It is noticed that skipping the DM stage results in an LKAS failure. This is because PR algorithms operate in the RGB domain and they do not work for the Bayer domain. So, DM is essential for proper LKAS operation. Additionally, it needs to be mentioned that certain approximation settings initially led to LKAS

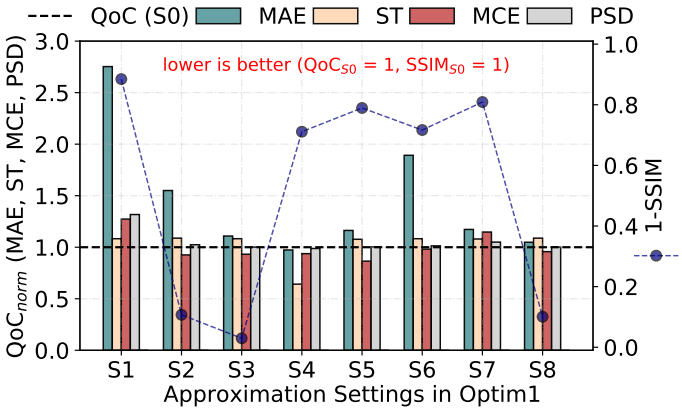



Figure 6.10: QoC compared with image quality for different approximation settings. LKAS is operated in approximation-only mode with default settings (Section 6.5.4).

failure. Minor modifications to the PR stage to handle these errors made it working (as further explained in Section 6.6.4).

Skipping one or more sub-stages in the ISP has both positive and negative impacts on the QoC of LKAS. The loss in image quality due to approximation settings (S1-S8) may degrade the QoC. However, the reduced sensing-to-actuation delay ($\tau$) allows faster sampling of the controller, thereby improving the QoC. Balancing this interaction is essential in determining if we gain or lose in final QoC. We evaluate the former by operating LKAS in approximation-only mode (without considering faster sampling) and the latter by operating LKAS in QoC-optimal mode (considering faster sampling).

Fig. 6.10 shows a comparison between image degradation and QoC of LKAS. The QoC metrics (MAE, ST, MCE and PSD) on the left x-axis are normalized to the accurate setting (S0). The SSIM loss (right y-axis) shows the degree of image degradation. First, it is evident that the different settings (S1-S8) have varying impact on image quality. S1 (skipping denoising DN) performs the worst in terms of both image quality and QoC. We conclude that skipping DN while keeping the rest of the stages (S1) is not a suitable candidate for getting better QoC. We also observe that some settings (S3, S4, S5, S7, S8) perform relatively similar to the baseline. This shows that ISP pipelines optimized for human vision are overkill for LKAS. There is no one-to-one correlation between image degradation and QoC. For instance, settings S4, S5 and S7 have high SSIM loss (more degradation) but they perform similar to S0 in terms of QoC, with S4 being slightly better. Contrarily, S2 has low SSIM loss but high MAE (worse QoC). This is explained by the fact that the performance of control algorithms depends on the presence of an essential feature in the image (lane markings in this case). Image-degradation metrics like SSIM loss fail to identify this. This non-correlation shows that approximating different

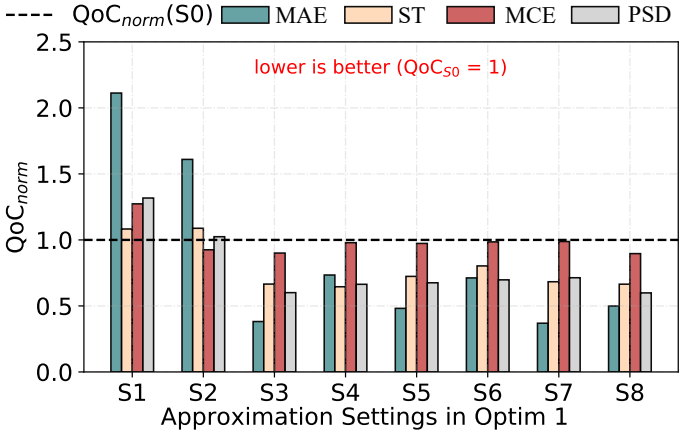


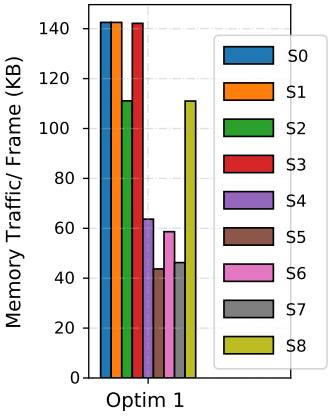

Figure 6.11: QoC improvements with reduced sampling period. LKAS is operated in QoC-optimal mode with default settings.

subsystems individually without considering the impact on the bigger closed-loop system can lead to sub-optimal results.

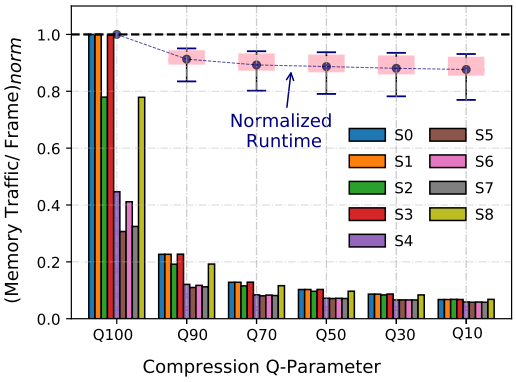
Fig. 6.9 shows that lowered compute workloads due to approximations (S1-S8) result in up to 76% reduction (S5) in the sensing-to-actuation delay ($\tau$). This allows faster sampling of the controller. This is taken into account in Fig. 6.11 that shows the QoC for the nine pipeline settings. We observe a significant impact of reduced sampling period on the QoC. Faster control sampling in S3-S8 due to reduced $\tau$ overshadows the negative impact of image degradation on QoC. We observe up to 63% (S7), 35% (S4), 10% (S8), 40% (S8) improvements in MAE, ST, MCE and PSD respectively. However, in settings S1 and S2, the minor reductions in $\tau$ do not allow a faster sampling. So, the negative impact of image degradation is not balanced, resulting in worsened QoC. From Fig. 6.10, we observe that skipping only gamut mapping (S3) and keeping only gamut mapping (S7) have opposite impacts on the visual quality of the image. However, both S3 and S7 have minimal effects on the essential features (lane markings) of the image [21]. This is evident in Fig. 6.11 where both settings S3 and S7 perform similarly in terms of QoC (0.38 and 0.36 respectively) with S7 being slightly better than S3. Fig. 6.12(a) reports the memory improvements in LKAS due to Optim 1. Up to 69% reduction (S5) in memory traffic is obtained from Optim 1.

### 6.6.3 Optim 2: lossy compression using variable Q-parameter

As a next step, we vary the Q-parameter in the JPEG algorithm to control the degree of lossy compression to reduce the off-chip memory traffic. In the evaluation of Optim 1, a fixed Q-parameter (= 100) was considered. For Optim 2, the




(a) Memory traffic reduction due to Optim 1.




(b) Off-chip memory traffic reduction and runtime improvements from variable Q-parameter (Optim 2).

Figure 6.12: Memory traffic reductions

Q-parameter is modulated in fixed steps from Q10 to Q90 in addition to Q100. Fig. 6.12(b) shows the off-chip memory traffic reduction for the different approximation settings while varying the Q-parameter. All values are normalized to $S0_{\text{Optim 1}}$. S5 is the most memory-efficient setting across all Q-parameters. For S5, additional memory reductions of up to 81% (Q10) are obtained over Optim 1. However, these reductions come at a cost; they introduce more noise to the system. To gain on QoC for the entire system, there must be significant runtime reductions in $\tau$ to allow for faster control sampling. We observe that for lower values of Q ($Q < 70$), the additional runtime reductions are insignificant (see Fig. 6.12(b)). Lower Q-parameter values lead to more aggressive quantization during compression. However, for smaller pixel values (more common in approximated images), values are rounded to zero resulting in no additional reduction in memory traffic, and thereby no runtime improvements. So, we consider only higher values of Q ($\geq 70$).

### 6.6.4 Interstage effects of approximations

It is observed that both the coarse-grained approximations in Optim 1 and the lossy compression in Optim 2, have a quality impact on the subsequent stages, especially the PR stage. However, the impact of Optim 1 on the PR stage is critical, as it results in LKAS failure in some cases. The PR stage does not detect the lanes properly for certain approximated streams obtained from S1-S8. The problem is traced back to the static image thresholding step in PR (see Fig. 6.3 (b)), which fails to identify the lane markers from the grayscale bird's eye view image due to incorrect choice of threshold. To counter this, Otsu's binarization algorithm is used, which dynamically identifies the optimal threshold. Otsu's algorithm brings an ad-

ditional computational complexity of $MN+7L^2+5L-12$, for an $M \times N$ image with $L$ grey levels. Depending on the approximation settings, dynamic thresholding is either performed on the grayscale bird's eye view image or RGB bird's eye view image. This results in desired LKAS behaviour across all approximation settings. It is noted that performing dynamic thresholding on the RGB image has thrice the computational complexity, which has been taken into account in our evaluation results.

## 6.7 Optim 3: Approximation-aware control design

To design a controller that is robust against approximation errors, we quantify the errors introduced due to coarse-grained approximations as well as variable lossy compression and use it to design an approximation-aware controller. Initially, we need to identify the system state parameter(s) affected by the approximations. For LKAS, the lateral deviation $y_L$ is affected by approximation. We quantify the error $(e_i)$ due to approximation for the setting S$i$ as the covariance of the calculated $y_L^i$ for the approximation setting S$i$ with respect to the calculated $y_L^0$ for the accurate setting S0, i.e., $e_i = \frac{1}{n}\sum_{j=1}^{n}(y_{L_j}^i - y_{L_j}^0)^2$.



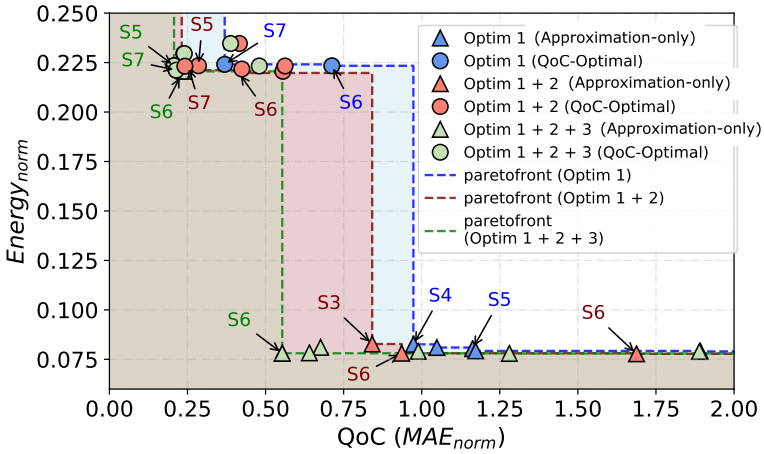



Figure 6.13: QoC-Energy trade-offs for Optim 3. Pareto-front improvements over Optim 1 and Optim 2 are shown. Note that the energy here is a measure of the degree of approximation in the approximation-only mode. The lower the energy, the higher the degree of approximation. The reader is referred to [36] for more details on the energy-aware optimisation.

We use the optimal LQG control design [48] technique to design the

approximation-aware controller for the system defined as follows:

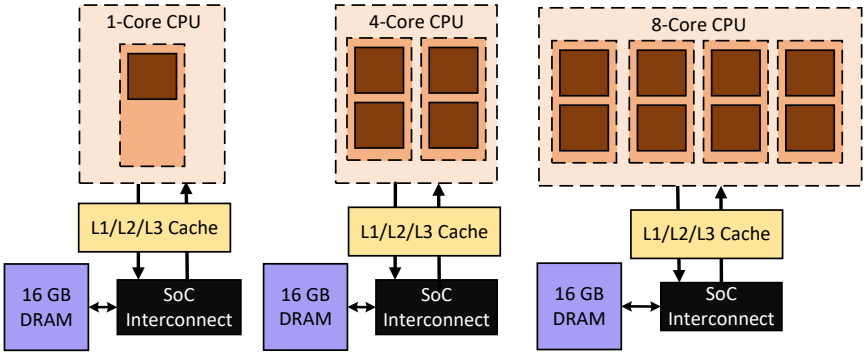
$$\dot{x}(t) = A_c x(t) + B_c u(t),$$
$$y(t) = C_c x(t) + e_i,$$

where $e_i$ models the error due to approximation as measurement noise for the output.

Fig. 6.13 shows the QoC-Energy trade-offs for Optim 3, approximation-aware LQG control, for the LKAS case study. Recall from Section 6.4.2 that the results so far were obtained by using an LQR controller. The control error of the LQR controllers for the various pipeline settings was quantified as explained above and used to design an LQG controller for each setting. We observe that the area under the Pareto curve for Optim 3 improves by 22% and 15% over Optim 1 and 2, respectively. This means that better trade-offs in terms of QoC and energy are obtained by the LQG controllers. It is important to mention that Optim 3 has QoC improvements over Optim 2 but no energy improvements, as it does not influence the compute-intensive sensing stage (S). This explains why the Pareto front only moves towards the left. Finally, the improvements from Optim 3 in approximation-only mode are higher than in QoC-optimal mode. This is because, in approximation-only mode, faster sampling is not considered. So, control decisions at each actuation are valid longer. So, better decisions taken by the LQG controller are more profound in approximation-only mode compared to QoC-optimal mode.



## 6.8 Approximation and mapping interplay




(a) CPU_1C (b) CPU_4C (c) CPU_8C

Figure 6.14: Different platform configurations considered in addition to Fig. 6.4 (a).

In Sections 6.6 and 6.7, we mapped all the approximate tasks to a default (CPU_8C+GPU) platform configuration (see Fig. 6.4 (a)). Mapping tasks to a GPU

has runtime benefits, but it is extremely power hungry. The number of CPUs online in the NVIDIA AGX Xavier platform is controlled using software-controlled power gating. We use this to introduce three new platform configurations (CPU_1C, CPU_4C, CPU_8C) in Fig. 6.14. We report our mapping results considering design points obtained by combined application of Optim 1, 2, and 3.

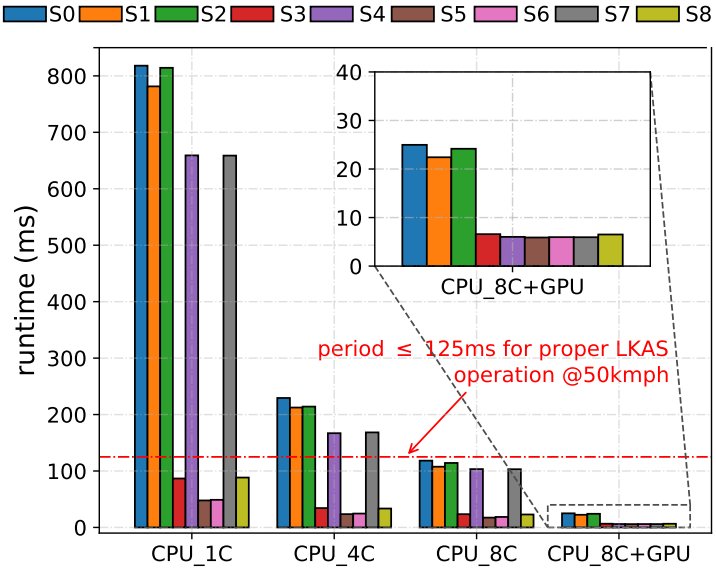



Figure 6.15: Comparative analysis of the runtime implications of the different platform mappings for all approximation settings S0-S8.

Fig. 6.15 shows a snapshot for the timing implications of different task mappings for all approximate settings S0-S8. Sensing-to-actuation delays $\tau$ are shown on the y-axis. We have observed in our experiments that for proper LKAS operation at a vehicle speed of 50 kmph, a controller sampling period $\leq$ 125 ms is required. So, mapping the accurate setting S0 to CPU_1C or CPU_4C results in a vehicle crash. However, mapping approximate settings S3, S5, S6 and S8 to CPU_1C and CPU_4C results in desired LKAS behaviour (in QoC-optimal mode). As mentioned earlier, a minimum frame rate of 8 fps (period $\leq$ 125 ms) is required for proper LKAS operation. In approximation-only mode, for each mapping, settings S1-S8 operate at the same frame rate as S0 of the corresponding mapping. Thus, all the settings result in LKAS failure for CPU_1C and CPU_4C. In QoC-Optimal mode, settings S3, S5, S6, S8 mapped to CPU_1C and CPU_4C result in desired behaviour due to improved sampling period. These results show that approximation enlarges the design space substantially, allowing the developer to find an appropriate balance between QoC, resource usage, and energy cost.

## 6.9 Sensitivity to environmental scenarios

IBC systems operate under different environmental scenarios. In this section, we evaluate the impact of approximation noise on IBC performance when operating under these scenarios. Fig. 6.16 shows the six different environmental scenarios considered for our evaluation. These are commonly encountered driving conditions relevant for LKAS.

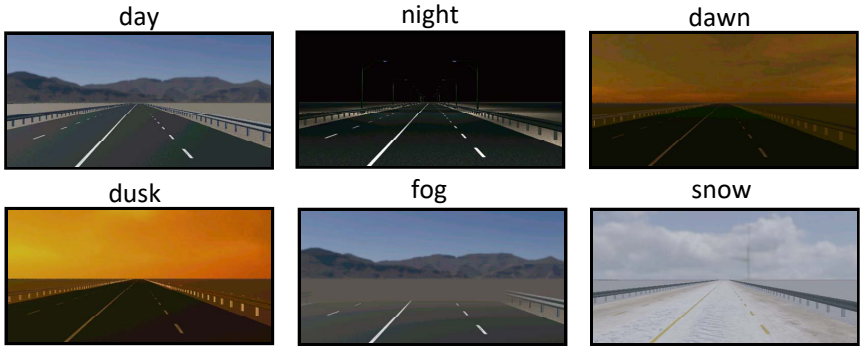



Figure 6.16: Different environmental scenarios considered in this work. The figures are obtained from the IMACS framework.

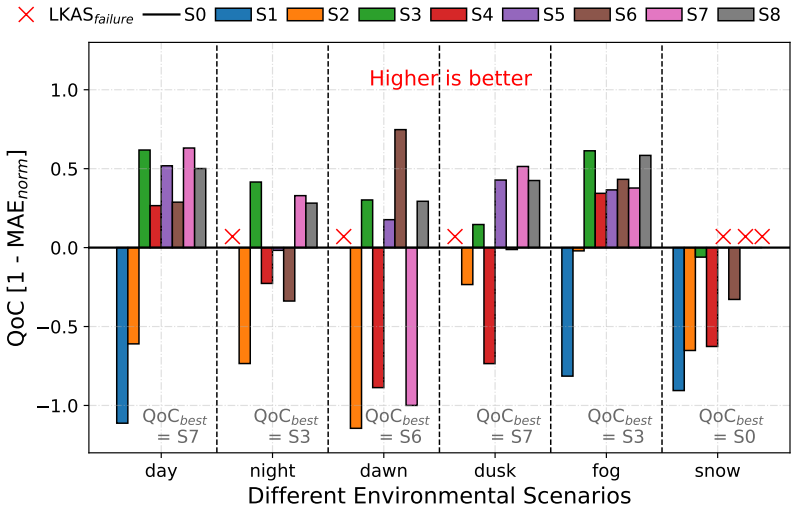



Figure 6.17: Sensitivity of QoC to different approximation settings operating under different environmental scenarios. Results are for Optim 1.

Fig. 6.17 shows the QoC sensitivity for LKAS to approximation settings (S0-S8) when operated under different environmental scenarios. All values are normal-

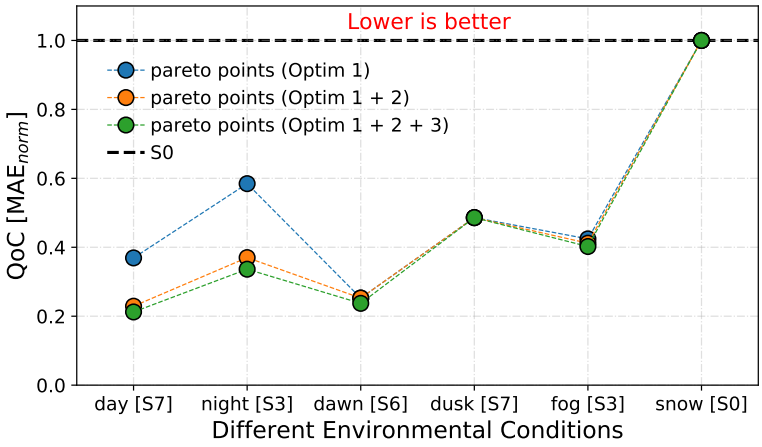



Figure 6.18: QoC improvements due to cross-layer optimizations (Optim 1, 2, 3) across different environmental scenarios.

ized to the S0 QoC (baseline). It is observed that the choice of approximation (S1-S8) is critical for better QoC. To get the best QoC, different approximation settings should be chosen based on the environmental scenario (grey markings in Fig. 6.17). S1 (skipping denoising) fails for night, dawn and dusk, while it performs worse than the baseline for the other scenarios. Similarly, S2 (skipping color map) performs worse than the baseline across all scenarios. This can be explained by the fact thatvthe sampling period for these cases is not improved compared to S0, while the added extra error makes the QoC worse. It is also observed that none of the approximation settings (S1-S8) improves over the baseline when operating in a snowy scenario. Settings S5, S7 and S8 lead to LKAS failure for this case. This is due to the lack of significant difference in pixel intensity between the lane markings and the road region. From this, we can conclude that the impact of approximation error on LKAS performance is highly sensitive to the operating environment. Dynamic selection of the approximation setting by recognizing the operating environmental scenario is required.

Fig. 6.18 shows the impact of the cross-layer optimizations on QoC for different scenarios. All the values are normalized to S0 for the corresponding scenario. For this analysis, we choose only the settings that give the best QoC per scenario in Fig. 6.17. Firstly, there is no improvement over S0 for snow as none of the approximation settings perform better than the baseline as explained earlier. For all other scenarios, Optim 1 gives QoC improvements over S0. For dawn, dusk and fog, we see no/minor incremental QoC improvements when we apply Optim 2 and 3 on top of Optim 1. This is because these are challenging cases for proper dynamic thresholding in the PR stage. When we add extra noise due to lossy compression, the performance of dynamic thresholding worsens. In case of dawn and

fog, the reduced $\tau$ due to Optim 2 and the approximation-aware controller of Optim 3 overpower the impact of worsened PR and we get slight QoC improvements over Optim 1. However, this is not the case in the dusk. Also, we observe higher QoC improvements for the night compared to the day. This is because, for the night, there is a higher difference in intensity between lane pixels and road pixels, compared to that of the day scenario. This results in a better PR performance, thus, larger QoC improvements.

We considered six different environmental scenarios most relevant to LKAS. It is worth noting that our approach is applicable to a combination of these scenarios as well. For instance, a segment of road with multiple tunnels can be handled by dynamically switching from a day to a night scenario and vice versa. Dynamic adaptation to environmental scenarios fits seamlessly in the SPADE design philosophy that is already scenario-driven.

## 6.10 Robustness of approximate IBC

IBC systems as considered in this work are safety-critical. This requires a robustness study when such a system is subjected to approximations. To this end, we evaluate the failure probability of LKAS when subjected to different approximate settings (S1-S8). First, we perform Monte-Carlo simulations of the entire system while taking into account different approximate settings and environmental scenarios. Then, we obtain the percentage of lane misprediction and closed-loop failure probability of LKAS.



### 6.10.1 Monte-Carlo simulation

For Monte-Carlo simulation, we sweep different LKAS parameters in Webots using the HiL setup (Section 6.3) and obtain various system models for simulation. The considered parameters are initial starting position or initial lateral deviation of the vehicle, different weather conditions and different road surface types. From these simulations, the camera frames obtained by driving the vehicle are extracted and stored as a dataset for further analysis. For each frame in the dataset, the lane markings are annotated as *ground truth* (information extracted from the accurate image).

### 6.10.2 Lane misprediction (LM)

Correct lane prediction is essential for proper LKAS operation. So, in this analysis, we study the sensitivity of the PR stage to different approximate settings by calculating the increase in lane misprediction (LM) compared to the predictions using the *ground truth*. We use the dataset obtained using Monte-Carlo simulations.

Lane misprediction (LM) is calculated as shown below [134]:

$$LM = 1 - \frac{1}{n}\sum_{i=1}^{n}\frac{PL_i}{G_i}$$

where $PL_i$ is the number of correctly predicted lane points per frame and $G_i$ is the number of *ground truth* points per frame. A prediction is considered correct if the difference between a *ground truth* point and the predicted point is less than a certain threshold. $n$ is the total number of frames considered. In this work, $G_i$ = 512 and $n$ = 3000. Fig. 6.19 shows that the PR stage is robust to errors from ap-

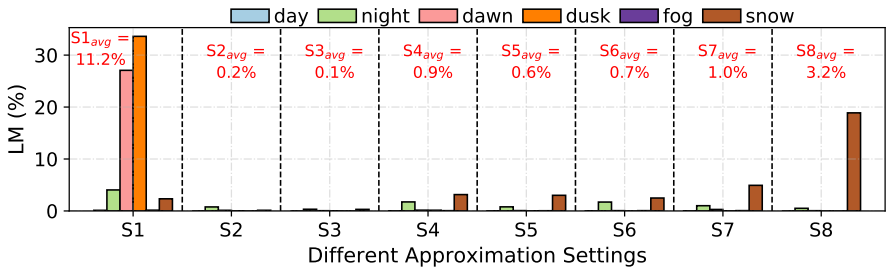



Figure 6.19: Sensitivity of PR to different approximate settings when operating under different environmental scenarios. The average LM for each setting across all scenarios is shown in red.

proximation settings S2-S7 with average LM ≤ 1%. We also see that image frames subjected to approximate settings S4-S7 have high visual changes (Fig. 6.10) compared to the accurate one, but they still have low LM. This is because essential lane markings are not affected by these visual changes. Skipping denoising (S1) has a high negative impact on lane detection (PR) with average LM = 11.7%. Thus, skipping denoising (S1) while keeping the rest of the stages is not suitable for proper LKAS performance. We also observe that keeping only tone mapping (S8) works for all scenarios except snow (LM = 18%) which shows that scenario-based approximation selection is needed.

### 6.10.3 Failure probability (FP)

From the results of the previous subsection, we determine the worst-case approximation error (Section 6.7) per setting considering the scenario with the highest LM for that setting. This error is used for designing different LQG designs for FP analysis. We then calculate the failure probability of LKAS for different approximate settings (S1-S8) using Monte-Carlo simulations of the entire system. There are two questions that we need to address here: (a) Is the designed controller stable and robust? (b) Is there LKAS failure, i.e, does the vehicle go out of the lane?

To evaluate stability robustness of the designed LQG controller, we calculate the stability radius ($r$) which is the radius of the largest ball centered at the $(-1,0)$ point to loop transfer function of the system in the nyquist plot [75]. $r$ is calculated as follows:

$$r = \frac{1}{||S||_\infty}$$

where $S$ is the sensitivity function calculated at the output of PR for the proposed LKAS model and $0 \leq r \leq 1$. A higher value of $r$ means more stability robustness to sensor noise. The $r$ values obtained for S0-S8 are 0.8787, 0.9800, 0.9129, 0.9906, 0.9900, 0.9982, 0.9898, 0.9982 and 0.9999, respectively. All the $r$ values are close to 1, which shows the stability of the designed LQG controllers. It is noted that the $r$ values for the approximate settings (S1-S8) are higher than the accurate (S0) one. This means that for the approximate cases, the LQG contoller is designed to sustain a higher sensor noise margin. This has impact on the LQG performance when evaluated in terms of FP.

Failure probability of the LKAS is calculated as shown below:

$$FP_{per\ km} = \frac{1}{m \times l} \sum_{i=1}^{m} F_i$$



where $F_i$ is 1 if the number of times the vehicle goes out of the lane markings during the simulation interval is one or more; otherwise, it is 0. $m$ is the number of Monte-Carlo simulations performed for each approximate setting. For our experiments, we consider $m$ = 50000. $l$ is the lifetime of the vehicle. The typical lifetime of a conventional vehicle is 150,000 mi or 241,401.6 km [17].

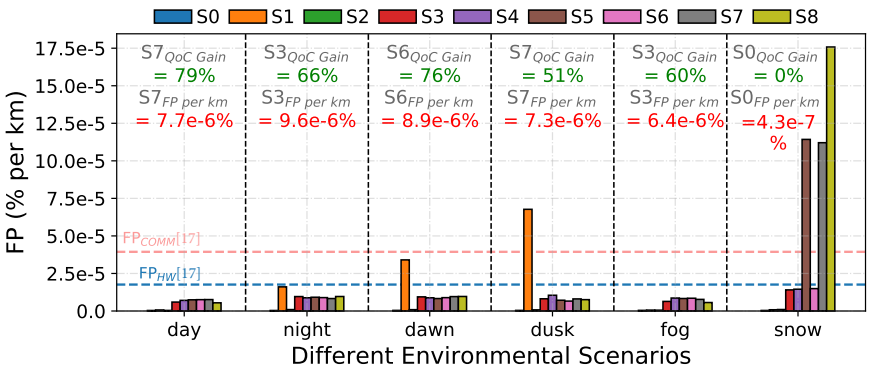



Figure 6.20: Failure probability of LKAS for different approximate settings when operating under different environmental scenarios. Failure probability of the best performing setting per scenario is highlighted in red.

Fig. 6.20 shows the FP of LKAS for different approximate settings under different environmental scenarios. To get some perspective, we plot the $FP_{per\ km}$ in the

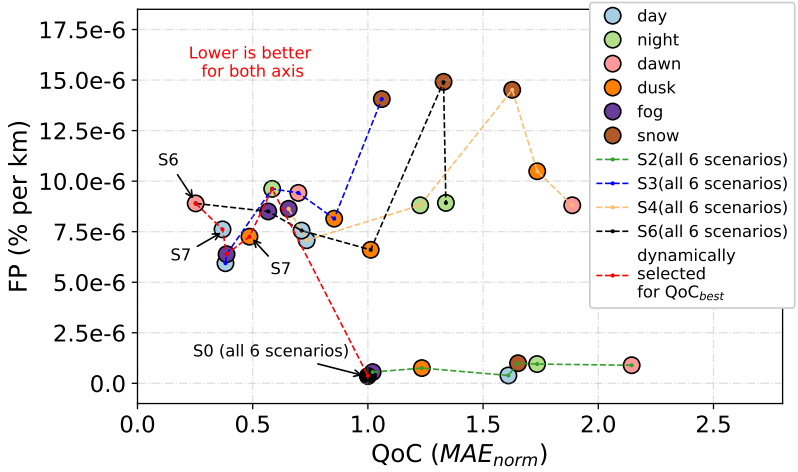



Figure 6.21: Comparative study showing LKAS performance (QoC) versus failure probability for the different approximate settings when operating under different environmental scenarios.

hardware and the communication subsystems of the vehicle as reported in [17]. We notice that approximate settings S2, S3, S4 and S6 have a low failure probability across all scenarios. Considering the best performing ($QoC_{best}$) setting per scenario, we notice that the system has worst-case $FP_{per\ km}$ of $9.6 \times 10^{-6}$% (S3 for the night) which is well below the failure probabilities of the hardware and the communication subsystems [5].

Considering FP in a standalone manner paints a partial picture. We are more interested in designs which not only have low FP but also perform better than the accurate setting (S0). Fig. 6.21 shows this comparative study between LKAS performance (QoC) and $FP_{per\ km}$. QoC values are normalized to the baseline (S0) for each scenario. S0 has the lowest FP as expected. We start by comparing the four static approximate settings S2, S3, S4 and S6 which have low FP across all scenarios. We observe that S2 has the lowest average FP compared to S3, S4 and S6, but its closed-loop QoC is worse than the baseline (S0) across all scenarios (see Fig. 6.21, green dotted line). Statically choosing S3, S4 and S6 across all scenarios shows improved QoC compared to baseline for some scenarios and degraded QoC for others (see Fig. 6.21, blue, yellow and black dotted lines). This motivates, as before, to dynamically select the approximate settings for each environmental scenario, which results in either improved or the same QoC as that of the baseline (S0). This is shown by the red dotted line in Fig. 6.21. Note that we choose (from

[5] Software, hardware and communication subsystems are responsible for the majority of the autonomous vehicle failures related to vehicular components as reported in [17]. Approximations proposed in this work contribute to software failure due to increased lane misprediction (LM). So, we report our results along with failure contributions due to discrepancies in hardware and communication subsystems to get a better perspective

left to right on the red dotted line in Fig. 6.21) S6 for dawn, S7 for day, S3 for fog, S7 for dusk, S3 for night and S0 for the snow environmental scenarios for the best QoC. A detailed analysis and the impact of switching between these scenarios at runtime is not in scope of this thesis. Interested readers are referred to [32].

The FP, in the scope of this chapter on approximation, may be used as a cut-off criterion for choosing the acceptable (Pareto-optimal) approximate settings for different environmental scenarios at runtime. From Fig. 6.20, we can observe that most settings for different scenarios satisfy the cut-off criteria $FP_{COMM}$ and $FP_{HW}$ from the literature. All the approximate settings that satisfy the FP criteria can be deployed at runtime. The next criterion for choosing the Pareto-optimal approximate setting for a particular environment scenario is then the QoC. The outcomes after the FP analysis from Fig. 6.21 are the same as the results in Section 6.9 (as illustrated in Fig. 6.18. This is as expected, given that the FP shows that all the optimal settings found in Section 6.9 satisfy the FP cut-off criterion. Note that in Fig. 6.21, the setting S3 has the lowest FP for the day scenario, lower than setting S7. Since both S3 and S7 can be deployed at runtime due to low LM, we choose S7 that has a better QoC compared to S3 for the day scenario.

## 6.11 Discussions

In this section, we provide a summary and give insights on some key aspects of our design approach in terms of dynamic configuration overheads, and applicability in safety-critical systems. We also discuss the generality as well as the modularity of our design approach while highlighting the ease of switching to newer design models.

### 6.11.1 Result summary and insights

In this section, we summarize the results presented in this chapter, looking from two different perspectives: (a) exploiting error-resilience in IBC systems and (b) sensitivity and robustness. A detailed list of parameters, optimizations and simulation settings considered for evaluation of our results is shown in Table 6.3.

**Exploiting error-resilience in IBC systems:** IBC systems like LKAS have intrinsic error resilience and visual quality is not paramount for such applications. We exploit this by performing coarse-grained computation skipping in the ISP as well as variable lossy compression post-ISP. This significantly reduces the sensing-to-actuation delay $\tau$ at the cost of loss in image quality. Reduced $\tau$ enables faster sampling of the controller which not only negates the negative impact of image degradation on QoC but in turn improves QoC as shown in Sections 6.6.2 and 6.6.3. We model this error as sensor noise in the approximation-aware LQG controller, which improves the overall QoC further as shown in Section 6.7. Reduced compute workloads due to approximate ISP and a higher degree of lossy compression gives significant memory improvement as well. In approximation-only mode, we

obtain 88% reduction in memory footprint and 44% improvements in QoC compared to accurate implementation (see Fig. 6.12(b), S6 in Q90). In QoC-optimal mode, we obtain 78% improvements in QoC and 89% reduction in memory footprint (see Fig. 6.12(b), S7 in Q90).

Table 6.3: Summarizing details of evaluation parameters considered.

| Sec. | Optimizations Considered | Platform Mappings | Environmental Scenarios | Controller Type |
|---|---|---|---|---|
| **6.6.2** | Optim 1 | CPU_-8C+GPU | day | LQR |
| **6.6.3** | Optim 1 + 2 | CPU_-8C+GPU | day | LQR |
| **6.7** | Optim 1 + 2 + 3 | CPU_-8C+GPU | day | LQG* |
| **6.8** | Optim 1 + 2 + 3 | CPU_-8C+GPU<br>CPU_8C<br>CPU_4C<br>CPU_1C | day | LQG* |
| **6.9** | Optim 1 + 2 + 3 | CPU_ 8C+GPU | day, night<br>dawn, dusk<br>fog, snow | LQG* |
| **6.10** | Optim 1 + 2 + 3 | CPU_ 8C+GPU | day, night<br>dawn, dusk<br>fog, snow | LQG* |

**Optim 1:** ISP approximation settings S0-S8
**Optim 2:** Variable lossy compression **Platform:** NVIDIA AGX Xavier
**Optim 3:** Approximation-aware control
**Webots Settings:** Sensor images resolution = 720p (with downsampling to 512 × 256), camera frame rate between 30fps, 60fps and 120fps, depending on the sampling period of controller, initial vehicle position = 15 cm from lane center, lane width = 3.25 m, webots simulation step = 1 ms, vehicle speed = 50 km/hr.
**LQR**: linear quadratic regulator, **LQG**: linear-quadratic-Gaussian
* Approximation-aware control

**Sensitivity and robustness:** For proper in-field deployment of approximate IBC systems, a wide range of commonly encountered environmental scenarios are tested in Section 6.9. It is observed that the choice of approximation is critical and different approximation settings should be chosen for different scenarios to get the best performance (see Fig. 6.17, 6.18). However, the goal is not only to get the best performance per scenario but also to design a robust IBC system with a low FP as explored in Section 6.10. We observe that settings S2, S3, S4 and S6 have a low FP across all scenarios (see Fig. 6.20). However, statically choosing S2, S3, S4

or S6 does not give us the best performance and motivates dynamically switching between settings based on the operating scenario (see Fig. 6.21).

Summing it all up, we observe that choosing S7 for day & dusk, S3 for night & fog, S6 for dawn and S0 for snow gives us the best balance between performance and robustness (see Fig. 6.21). For this solution, we obtain average improvements of 51% in QoC, and 83% in memory, with the worst-case FP (per km) of $9.6 \times 10^{-6}$%. Thus, our design approach provides robust approximate IBC designs (with FP below those of the hardware and communication subsystems) with significant QoC and memory improvements.

### 6.11.2 Overhead of dynamic approximation selection

In Sections 6.9 and Section 6.10, we highlight the benefits of dynamic selection of approximation settings for different environmental scenarios. In this section, we discuss the additional overheads of such an approach. First, we classify the different environmental scenarios using a state-of-the-art convolutional neural network (CNN) classifier, ResNet-50 [58]. ResNet-50 has a classification accuracy of 94.75% on the ImageNet 2012 classification dataset that consists of 1000 classes with 1.28 million training images, 50k validation images and 100k test images [58]. We choose ResNet-50 pre-trained on ImageNet and train it on our dataset (see Section 6.10) using transfer learning. On our dataset, it achieves a classification accuracy of 99.72%. The higher classification accuracy is due to the fewer number of output classes compared to ImageNet (six in this work). ResNet-50 has a runtime penalty of 1.5 ms on the NVIDIA AGX Xavier [61], which is 6% of the overall runtime of LKAS (considering S0).

The proposed scenario classifier is invoked every frame to perform classification (every 25 ms for S0-S2, every 8.3 ms for S3-S8). But in real driving conditions, the transition between different weather scenarios is less frequent. So, we believe that the frequency at which the scenario classifier is invoked can be relaxed, thus, reducing the overhead of dynamic selection. Investigating this is out of the scope of this work. Moreover, the runtime overhead of the scenario classifier can be reduced by performing latency hiding. The scenario classifier can be scheduled on the DLA in parallel with the PR stage to get a lower sensor-to-actuation delay ($\tau$).

### 6.11.3 Approximations in safety-critical systems

We consider a LKAS, which is a safety-critical system. Approximations are not intuitive solutions in safety-critical systems. But recent efforts like NVIDIA PilotNet [18, 19] and LaneNet [96] have shown that neural network-based algorithmic approximation approaches can be used for safety-critical systems. In this work, we show that compute- and data-centric approximation approaches can be applied to a safety-critical system like LKAS when it comes with in-depth analysis and proper evaluation of the system-level QoC and FP. We show that the quality of the overall system improves despite errors being introduced in the intermediate

sub-systems. Additionally, failure probability analysis of the approximate system shows that failure rates are bounded within acceptable limits [17].

### 6.11.4 Generality and modularity

**Generality:** The proposed work applies to any feedback control system with camera-based sensing. These systems have massive compute workload, which can be reduced by approximating the sensing stage, as shown in this work. In essence, approximations improve the timing and memory simultaneously at the cost of sensor noise. However, this additional sensor noise can be tolerated due to the inherent error resilience of closed-loop IBC systems, thus, improving the overall QoC. Although the gains reported in this work are application- and platform-specific, the general idea is applicable to other IBC systems [16] like automatic pedestrian detection, vision-based predictive suspension systems and so on.
**Modularity:** This work demonstrates that approximating the compute-intensive sensing stage (S) in an IBC system provides significant QoC and memory benefits. Computation-skipping is explored in this work, which is one of the potential solutions for approximating S. Other techniques such as deep neural network (DNN) models can also be explored in future for replacing these compute-intensive sensing stages. They can be applied to replace either the ISP stage [62] or the PR stage [96] or both. To easily facilitate model changes, we have developed our framework in a modular fashion. DNNs can be approximated through quantization, pruning and other standard DNN optimisation techniques [85]. To integrate DNN-based approximation in our framework, we just need the execution time of the DNN model (in inference mode) on the considered hardware platform/inference engine. This parameter is used for designing approximation-aware control.



## 6.12 Conclusions

We have shown that compute-centric (ISP stage skipping) and data-centric (variable lossy compression) approximations are promising strategies for simultaneously optimizing QoC and memory usage of IBC systems. We have shown with extensive experiments that approximation-aware control designs are suitable when we take into account the artefacts of approximation in closed-loop systems while optimizing for QoC. Our design approach is shown to be applicable to a wide range of environmental scenarios. The best results are obtained when switching between different controller configurations (system scenarios) when environmental conditions change. We have also shown that approximate IBC systems designed using our approach are robust with a failure probability (FP per km) of $9.6 \times 10^{-6}$%. The presented approximation approach is shown to have high potential in a LKAS. Approximating the sensing task is complementary to the parallelisation and pipelining discussed in earlier chapters. The dynamic switching between environmental scenarios fits well in the scenario-based SPADE approach.

7

# Conclusions and Future Work

Multiprocessor image-based control (IBC) systems will only become more prominent and visible in the upcoming future. The prime driving factors for adopting multiprocessor IBC systems are the advances in the semiconductor industry, and the ease and necessity of connectivity, e.g., using 5G (and even 6G), for the modern industrial systems, devices and utilities. The advances in the semiconductor industry are driving down the price and size (form factor) of multiprocessors and high quality complementary metal–oxide semiconductor (CMOS) cameras. A connected world improves the overall efficiency and productivity of modern systems and the connected humans. Seamless connectivity enables data-intensive processing in the cloud and allows the integration of camera sensors as edge devices in modern industrial systems. A camera can sense multiple physical parameters and can be (re-)configured with only a software update. This thesis explores how we can effectively optimise multiprocessor IBC systems implementation and addresses some of the fundamental challenges with its adoption in industrial systems.



## 7.1 Conclusions

Multiprocessor IBC systems are a class of data-intensive feedback control systems whose feedback is provided by image-based sensing using cameras as sensors and implemented on a multiprocessor platform. Multiprocessor IBC systems have become popular with the advent of efficient image-processing algorithms, low-cost CMOS cameras with high resolution and low-cost multiprocessor platforms. The combination of the camera and the image-processing algorithm gives necessary information on parameters such as relative position, geometry, relative distance, depth perception and tracking of the object-of-interest. This enables the effective use of low-cost camera sensors to enable new functionality or replace expensive sensors in cost-sensitive industries like automotive and automation. Applications of multiprocessor IBC systems are found in robotics, autonomous vehicles, advanced driver assistance system (ADAS), electron microscopes, visual navigation and so on.

The challenge, however, is that the image-processing algorithms are compute-intensive and result in an inherent relatively long sensing delay. State-of-the-art

design methods do not fully exploit the IBC system characteristics and advantages of the multiprocessor platforms for optimising the sensing delay. The sensing delay of an IBC system is moreover variable with a significant degree of variation between the best-case and worst-case delay due to application-specific image-processing workload variations and the impact of platform resources. A long variable sensing delay degrades system performance and stability. A tight predictable sensing delay is required to optimise the IBC system performance and to guarantee the stability of the IBC system. Analytical computation of sensing delay is often pessimistic due to image-dependent workload variations or challenging platform timing analysis. Therefore, this thesis explores techniques to cope with the long variable sensing delay by considering application-specific IBC system characteristics and exploiting the benefits of the multiprocessor platforms. Effectively handling the long variable sensing delay helps to optimise IBC system performance while guaranteeing IBC system stability.

In the first contribution, the scenario- and platform-aware design (SPADE) flow is proposed that relates the formal dataflow timing analysis with control timing parameters for the formal controller design. This thesis proposes the first formal model-based design framework for the IBC system co-design that relates dataflow analysis and controller design. The SPADE flow brings together dataflow and control formalisms in the same framework. This relation allows to bring in the optimisation techniques from the dataflow domain into the control timing parameter optimisation. The first contribution also examines the case of application parallelism with no pipelining allowed for the control loop and thereby, reduces the sensor-to-actuator delay and period of the resulting controller implementation. The proposed SPADE flow also makes a step forward towards real-life implementation by detailing how the flow can be adapted for industrial platforms. Both academic and industrial platforms could implement the SPADE approach.



The second contribution integrates the parameters relevant for a practical pipelined implementation. A pipelined implementation reduces the sampling period while keeping the sensor-to-actuator delay constant. Inter-frame dependencies are present in many modern computer-vision algorithms. Such algorithms, when used in an IBC setting, require to model such algorithmic artefact at the control level. Moreover, system nonlinearities and constraints on system variables are other common factors encountered in almost all real-life settings. This thesis explicitly considers application-specific inter-frame dependencies and their impact on the controller implementation. This thesis presents a model-predictive control (MPC) formulation for pipelined IBC systems considering workload variations, inter-frame dependencies, system nonlinearities and constraints on system variables. This thesis, thus, makes a step towards a real-life pipelining implementation for IBC systems.

The third contribution, in essence, brings together the advantages of the above two contributions in the SPADE flow for pipelined parallelism. Pipelined parallelism effectively reduces both the sampling period and sensor-to-actuator delay while taking into account both the degree of application parallelism (determined

by the algorithm and the given platform allocation) and the degree of pipelining (quantified by inter-frame dependencies and the given platform allocation). An algorithm is detailed for the SPADE flow with modular blocks for binding and scheduling, controller design and timing analysis. These blocks are modular as any state-of-the-art technique could be used instead of those used in this thesis: i) the SDF3 tool for binding and scheduling; ii) the linear quadratic regulator (LQR), linear-quadratic-integral (LQI), linear-quadratic-Gaussian (LQG), MPC, and Markovian jump linear system (MJLS) controller design techniques; and iii) max-plus algebra for timing analysis. This thesis presents the required model transformations for realizing SPADE using the scenario-aware dataflow (SADF) model-of-computation (MOC). This thesis also details the adaptation of the SPADE flow for industrial platforms for pipelined parallelism.

The fourth contribution explicitly takes into account the application-specific workload variations and particularly considers the switching probabilities between the workload scenarios. Considering these workload variations and switching probabilities implies that we can reduce the average sensor-to-actuator delay and the average sampling period of the controller implementation. In literature, variable sensor-to-actuator delay (resulting from the variable workload) was dealt with through switched linear controllers with either known or unknown sequences of delay occurrences. These solutions either suffer from poor performance (from unknown arbitrary delay sequences) or unrealistic assumptions (when knowledge of the delay sequence is not available in reality). This thesis proposes an alternative controller design method based on the MJLS formulation. The image-workload variations are identified and modelled as a discrete-time Markov chain (DTMC), where each Markov state represents a workload scenario. At runtime, the IBC system switches between workload scenarios based on image workload. Having too many switching workload scenarios results in an unstable system or degrades system performance. This thesis thus proposes system-scenario identification that abstracts multiple workload scenarios based on camera frame rate, sensing delay and sampling period into system scenarios. The DTMC is then recomputed, considering only the system scenarios, and the controller is synthesised based on the MJLS formulation.

The fifth contribution explores approximate computing as a means to reduce the effective sensor-to-actuator delay and the sampling period. Approximate computing trades off accuracy in the signal processing for gains in response time. Approximation is in that sense complementary to parallelizing and pipelining the control loop. This thesis also presents the performance evaluation for IMAge-based Control Systems (IMACS) framework for analysing and validating the impact of injecting errors in the image processing on the closed-loop IBC system performance. In addition, this thesis proposes an approximation-aware control design that takes as input the quantified error due to approximation.

In conclusion, this thesis aims to efficiently cope with the long variable sensing delay of IBC systems so that engineers can deploy IBC systems efficiently in time- and safety-critical domains. An IBC system uses computer-vision algorithms for

sensing and control laws and algorithms for control regulation. The performance of the IBC system is dependent on the performance of both the sensing and control algorithms. The presented SPADE flow brings together two modeling and analysis paradigms in an integral way – the dataflow formalism and control theory. As a means for optimization, both platform-specific aspects and application-specific characteristics are considered in various contributions presented in this thesis. This thesis also describes the way to adapt the SPADE flow for both custom-made and industrial platforms, by considering several relevant design aspects such as image-workload variations, inter-frame dependencies, system nonlinearities and so on.

## 7.2 Future work

This thesis proposes the scenario- and platform-aware design for multiprocessor IBC systems implementation for an efficient model-driven optimisation. There are various ways to extend the SPADE flow for completeness and adoption in practice, which are interesting for future work. In the following, we present a few major future directions.

- **Automated model extraction:** Extracting the application and platform models automatically from algorithmic descriptions of the application and a given platform allocation is an important future direction to enable a wider usage of formal model-based design and an analysis framework such as SPADE. The model extraction is often tricky and error-prone in many real-life scenarios. A model-based design approach thrives on the accuracy of the inherent models. The current SPADE approach assumes that the application graph and the platform graph are given or modelled by the designer. An automated approach to extract these models for a given algorithmic implementation and for a given platform allocation is helpful for adoptability of the proposed model-based approaches in industry and avoiding manual efforts/errors in the modelling.

- **Analytical worst-case execution time (WCET) computation:** The application modelling in the SPADE flow using SADF requires the WCET of tasks (actors in the SADF model). For industrial platforms, the WCET analysis of tasks running on the platform is non-trivial. An adaptation of how runtime profiling can be used for the proposed SPADE flow is detailed in this thesis. The worst-case system scenario is identified (see Chapter 4) based on the profiled WCET and may not be the actual WCET. Computing the analytical WCET bound for tasks running on an industrial platform is hard and still an open problem. A tighter WCET bound is a helpful addition for the industrial adoption of the SPADE flow and can potentially improve the performance of the overall design. An exact WCET bound, typically obtained from pre-

dictable platforms, can maximise the system performance when using the SPADE approach.

- **Scalablility:** Scalability of the current SPADE flow is dependent on the SADF analysis and mapping (using the SDF3 tool in this thesis). As the size of the input SADF model grows, state-space explosion may be encountered during the analysis and mapping depending on the algorithms used for sensing. In particular, a meaningful addition to the framework is the ability to handle and treat deep neural networks (DNNs)-based sensing as an alternative to classical computer-vision algorithms used in this thesis. Given the wide-spread use of DNNs in various domains these days, it is a natural step forward where the challenge of long processing (inference) latency is even more valid. Typically, DNNs are huge and using them directly in the dataflow analysis is challenging and may lead to state-space explosion. A possible direction of future work is to consider optimisation techniques for dataflow analysis and mapping with constraints on latency of the SADF model along with the throughput-constraint, i.e., latency- and throughput-aware binding and scheduling.

- **Code generation:** The SPADE flow does currently not offer a code-generation capability. An industrial model-based design flow typically expects code-generation functionality to be embedded within the framework. Integrating a code-generation capability for the implementation is another interesting and useful direction for future work.



In addition, the following future directions would make the SPADE flow more complete without adding fundamentally new features.

- A SPADE adaptation for data-/compute-intensive control compute tasks (e.g., a compute-intensive MPC or path planning) is another interesting direction to explore for future work. In this thesis, the assumption is that the control compute and actuation tasks are not compute-intensive. Another implicit assumption is that the sensing task output is available before the control compute task can start. A compute-intensive control task implies that we can parallelise the control task and potentially even pipeline its execution. For a non-pipelined implementation, the sensing and control tasks are sequential and hence the compute-intensive control compute task results in longer delays and optimal control strategies are required for addressing the longer delay. For a pipelined implementation, the SPADE flow considers a constant sampling period and as fast as possible sensing. In this choice of control design, the control compute task uses the latest sensing data available at the start of its execution. If the control compute task is unnecessarily delayed, some sensing data needs to wait longer before it is processed by the control task. Hence, the control compute task needs priority for mapping and scheduling. The interplay between a compute-intensive

control task and the data-intensive sensing task needs to be explored carefully. The SPADE flow can still be used, but the design-space exploration (DSE) needs further scrutiny.

- Considering multiple applications including one or more IBC applications sharing a platform is not explored in this thesis in detail. The current SPADE flow takes a single IBC application as input. Multiple input applications give additional optimisation options for mapping, controller design and scheduling. A careful consideration of these optimisation options is needed for scalability and performance. The SPADE flow extension for considering multiple (IBC) applications is an interesting future work.

- The case study considered for demonstrating the current SPADE approach assumes a single-input single-output (SISO) control system. Although the SPADE theory allows in principle a design for multi-input multi-output (MIMO) control systems, the technical challenges still need to be addressed. A MIMO system, for instance, could have multiple sensors. This could mean that the information is coming at multiple rates. A control mechanism to deal with these multiple rates could also have implementation constraints. And the mapping design space is enlarged. This is again an interesting direction for future work.

- In this thesis, we have shown how multiple controller design methods can be integrated with the SPADE approach. However, a rigorous analysis on identifying the controller design technique that is optimal for a given application and platform has not been done. This is another interesting direction for future work.

# Acronyms

**ADAS** advanced driver assistance system.

**AEB** automatic emergency braking.

**AI** artificial intelligence.

**ARX** auto-regressive exogenous.

**ASIL** automotive safety integrity level.

**CCD** charge-coupled device.

**CMOS** complementary metal–oxide semiconductor.

**CNN** convolutional neural network.

**CoG** centre of gravity.

**CompSOC** composable and predictable multiprocessor system-on-chip.

**CPS** cyber-physical system.

**CPU** central processing unit.

**CQLF** common quadratic Lyapunov function.

**dGPU** discrete GPU.

**DNN** deep neural network.

**DRAM** dynamic random access memory.

**DSE** design-space exploration.

**DTMC** discrete-time Markov chain.

**FP** failure probability.

**FPS** frames per second.

**FSD** full self-driving.

**GM** gain margin.

**GMSL** gigabit multimedia serial link.

**GPU** graphical processing unit.

**HIL** hardware-in-the-loop.

**HSDFG** homogeneous synchronous dataflow graph.

**I/O** input/output.

**IBC** image-based control.

**iGPU** integrated Pascal GPU.

**IMACS** performance evaluation for IMAge-based Control Systems.

**ISP** image-signal processing.

**LKAS** lane-keeping assist system.

**LMI** linear matrix inequality.

**LPV** linear parameter-varying.

**LQG** linear-quadratic-Gaussian.

**LQI** linear-quadratic-integral.

**LQR** linear quadratic regulator.

**LTI** linear time-invariant.

**LUT** look-up table.

**MAE** mean absolute error.

**MCE** maximum control effort.

**MIMO** multi-input multi-output.

**MJLS** Markovian jump linear system.

**MoC** model-of-computation.

**MPC** model-predictive control.

**MPSoC** multiprocessor system-on-chip.

**MSE** mean square error.

**NoC** network-on-chip.

**NPU** neural processing unit.

**ODT** object detection and tracking.

**OS** operating system.

**PM** phase margin.

**PR** perception.

**PSD** power spectral density.

**QoC** quality-of-control.

**QP** quadratic programming.

**RMSE** root mean square error.

**RoI** region-of-interest.

**SADF** scenario-aware dataflow.

**SDF** synchronous dataflow.

**SDFG** synchronous dataflow graph.

**SiL** software-in-the-loop.

**SISO** single-input single-output.

**SLC** switched linear control system.

**SLR** single-lens reflex.

**SoC** system-on-chip.

**SPADe** scenario- and platform-aware design.

**SSIM** structural similarity.

**ST** settling time.

**TCP/IP** transmission control protocol/internet protocol.

**TDM** time-division multiplexing.

**WCET** worst-case execution time.

**XIL** X-in-the-loop.

**ZOH** zero-order hold.

# List of Figures

# List of Tables

# Bibliography


[1] Shreya Adyanthaya, Zhihui Zhang, Marc Geilen, Jeroen Voeten, Twan Basten, and Ramon Schiffelers. Robustness analysis of multiprocessor schedules. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2014. (Cited on pages 7, 26, 74, 117, 122, 125, and 131.)

[2] Rahul Agrawal, Soumyajit Gupta, Jayanta Mukherjee, and Ritwik Kumar Layek. A gpu based real-time cuda implementation for obtaining visual saliency. In *Proceedings of the Indian Conference on Computer Vision Graphics and Image Processing*, 2014. (Cited on page 11.)

[3] Jim Al-Khalili. In retrospect: Book of optics. *Nature*, 518(7538):164–165, 2015. (Cited on page 2.)

[4] Hadi Alizadeh Ara, Amir Behrouzian, Martijn Hendriks, Marc Geilen, Dip Goswami, and Twan Basten. Scalable analysis for multi-scale dataflow models. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(4), 2018. (Cited on pages 31, 33, 34, 84, and 86.)

[5] Saoussen Anssi, Karsten Albers, Matthias Dörfel, and Sébastien Gérard. chronVAL/chronSIM: a tool suite for timing verification of automotive applications. In *Embedded Real Time Software and Systems (ERTS)*, 2012. (Cited on page 76.)

[6] AR1335 CMOS Digital Image Sensor for Automotive Applications. `https://www.onsemi.com/products/sensors/image-sensors-processors/image-sensors/ar1335`. [Online; accessed 29-Dec-2021]. (Cited on pages 116 and 141.)

[7] K-E Arzén, Anton Cervin, Johan Eker, and Lui Sha. An introduction to control and scheduling co-design. In *39th IEEE Conference on Decision and Control (CDC)*, volume 5, 2000. (Cited on page 10.)

[8] Karl J Åström and Björn Wittenmark. Computer-controlled systems: Theory and design. *Courier Corporation*, 2013. (Cited on pages 7, 59, 64, and 74.)

[9] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. Synchronization and linearity: an algebra for discrete event systems. *John Wiley & Sons Ltd*, 1992. (Cited on pages 33 and 84.)

[10] Martin Becker, Sajid Mohamed, Karsten Albers, Partha Pratim Chakrabarti, Samarjit Chakraborty, Pallab Dasgupta, Soumyajit Dey, and Ravindra Metta. Timing analysis of safety-critical automotive software: The AUTOSAFE tool flow. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 385–392. IEEE, 2015. (Cited on page 207.)

[11] Jan Behmann, Kelvin Acebron, Dzhaner Emin, Simon Bennertz, Shizue Matsubara, Stefan Thomas, David Bohnenkamp, Matheus T Kuska, Jouni Jussila, Harri Salo, et al. Specim iq: evaluation of a new, miniaturized handheld hyperspectral camera and its application for plant phenotyping and disease detection. *Sensors*, 18(2):441, 2018. (Cited on page 4.)

[12] Ron Bell. Introduction to iec 61508. In *ACM International Conference Proceeding Series*, volume 162, pages 3–12. Citeseer, 2006. (Cited on page 5.)

[13] Alberto Bemporad, Daniele Bernardini, Michael Livshiz, and Bharath Pattipati. Supervisory model predictive control of a powertrain with a continuously variable transmission. *SAE Technical Paper 2018-01-0860*, 2018. (Cited on page 57.)

[14] Alberto Bemporad, Daniele Bernardini, Ruixing Long, and Julian Verdejo. Model predictive control of turbocharged gasoline engines for mass production. *SAE Technical Paper 2018-01-0875*, 2018. (Cited on page 57.)

[15] Alberto Bemporad, Francesco Borrelli, Manfred Morari, et al. Model predictive control based on linear programming-the explicit solution. *IEEE transactions on automatic control*, 47(12):1974–1985, 2002. (Cited on page 124.)

[16] Klaus Bengler, Klaus Dietmayer, Berthold Farber, Markus Maurer, Christoph Stiller, and Hermann Winner. Three decades of driver assistance systems: Review and future perspectives. *IEEE Intelligent Transportation Systems Magazine*, 6(4), 2014. (Cited on pages 5, 24, 136, and 167.)

[17] Parth Bhavsar, Plaban Das, Matthew Paugh, Kakan Dey, and Mashrur Chowdhury. Risk analysis of autonomous vehicles in mixed traffic streams. *Transportation Research Record*, 2625(1):51–61, 2017. (Cited on pages 138, 162, 163, and 167.)

[18] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. (Cited on page 166.)

[19] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017. (Cited on page 166.)

[20] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017. (Cited on page 65.)

[21] Mark Buckler, Suren Jayasuriya, and Adrian Sampson. Reconfiguring the imaging pipeline for computer vision. In *IEEE International Conference on Computer Vision (ICCV)*, pages 975–984, Oct 2017. (Cited on pages 11, 46, 139, 140, 143, 146, and 153.)

[22] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013. (Cited on page 5.)

[23] Camera. `https://en.wikipedia.org/wiki/Camera`. [In Wikipedia. Online; accessed 29-Dec-2021]. (Cited on page 2.)

[24] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and K-E Arzen. How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime. *IEEE control systems magazine*, 23(3), 2003. (Cited on page 10.)

[25] Indrasis Chakraborty, Siddhartha S Mehta, J Willard Curtis, and Warren E Dixon. Compensating for time-varying input and state delays inherent to image-based control systems. In *American Control Conference (ACC)*, 2016. (Cited on pages 5 and 24.)

[26] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Design, Automation & Test in Europe Conference (DATE)*, 2003. (Cited on page 10.)

[27] Arun Chandrasekharan, Daniel Große, and Rolf Drechsler. Proact: A processor for high performance on-demand approximate computing. In *Proceedings of the on Great Lakes Symposium on VLSI*, GLSVLSI '17, page 463–466, New York, NY, USA, 2017. Association for Computing Machinery. (Cited on pages 11 and 139.)

[28] Marieke BG Cloosterman, Nathan Van de Wouw, WPMH Heemels, and Hendrik Nijmeijer. Stability of networked control systems with uncertain time-varying delays. *IEEE Transactions on Automatic Control (TACON)*, 54(7), 2009. (Cited on page 107.)

[29] Peter Corke. Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised. *Springer*, 118, 2017. (Cited on pages 5, 6, 24, 73, and 122.)

[30] Oswaldo Luiz Valle Costa, Marcelo Dutra Fragoso, and Ricardo Paulino Marques. *Discrete-time Markov jump linear systems*. Springer, 2006. (Cited on pages 123, 126, 127, and 128.)

[31] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3), 2007. (Cited on page 10.)

[32] Sayandip De, Yingkai Huang, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Hardware- and situation-aware sensing for robust closed-loop control systems. In *Design, Automation and Test in Europe Conference (DATE)*, 2021. (Cited on pages 164 and 207.)

[33] Sayandip De, Jos Huisken, and Henk Corporaal. Designing energy efficient approximate multipliers for neural acceleration. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 288–295, Aug 2018. (Cited on pages 11, 139, and 146.)

[34] Sayandip De, Jos Huisken, and Henk Corporaal. An automated approximation methodology for arithmetic circuits. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019. (Cited on pages 11 and 139.)

[35] Sayandip De, Sajid Mohamed, Konstantinos Bimpisidis, Dip Goswami, Twan Basten, and Henk Corporaal. Approximation trade offs in an image-based control system. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1680–1685. IEEE, 2020. (Cited on pages 18, 118, 135, 182, and 206.)

[36] Sayandip De, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 8: 174568–174586, 2020. (Cited on pages 16, 18, 110, 135, 136, 155, 183, and 206.)

[37] Richard C Dorf and Robert H Bishop. Modern control systems. *Pearson*, 2011. (Cited on pages 29, 42, 55, 87, and 122.)

[38] Stephen A Edwards and Edward A Lee. The case for the precision timed (PRET) machine. In *Design Automation Conference (DAC)*. IEEE, 2007. (Cited on page 26.)

[39] Abbas El Gamal and Helmy Eltoukhy. Cmos image sensors. *IEEE Circuits and Devices Magazine*, 21(3):6–20, 2005. (Cited on page 3.)

[40] Jos Elfring, Rein Appeldoorn, Sjoerd van den Dries, and Maurice Kwakkernaat. Effective world modeling: Multisensor data fusion methodology for automated driving. *Sensors*, 16(10), 2016. (Cited on pages 5 and 24.)

[41] Ashok Kumar Elluswamy, Matthew Bauch, Christopher Payne, Andrej Karpathy, Dhaval Shroff, Arvind Ramanandan, and James Robert Howard Hakewill. Predicting three-dimensional features for autonomous driving, October 19 2021. US Patent 11,150,664. (Cited on page 4.)

[42] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460, Dec 2012. (Cited on pages 11, 139, and 146.)

[43] James Estrin. Kodak's First Digital Moment. `https://lens.blogs.nytimes.com/2015/08/12/kodaks-first-digital-moment/`, 2015. [Online; accessed 29-Dec-2021]. (Cited on page 3.)

[44] FEI. An introduction to electron microscopy. *ISBN:978-0-578-06276-1*, 2010. (Cited on pages 5 and 24.)

[45] Sarah Feldman. Picture Perfect: The Evolution of the Camera. `https://www.statista.com/chart/18488/camera-timeline/`, 2019. [Online; accessed 29-Dec-2021]. (Cited on pages 2 and 179.)

[46] Daniele Fontantelli, Luigi Palopoli, and Luca Greco. Optimal cpu allocation to a set of control tasks with soft real–time execution constraints. In *16th International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2013. (Cited on pages 11, 56, 70, 71, and 114.)

[47] Dustin Franklin. NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics. In *NVIDIA Developer Blog*, 2018. (Cited on pages 8, 116, 136, 141, 144, and 183.)

[48] Gene F Franklin, J David Powell, Michael L Workman, et al. *Digital control of dynamic systems*, volume 3. Addison-wesley Menlo Park, CA, 1998. (Cited on pages 143, 144, and 155.)

[49] Hiroshi Fujimoto. Visual servoing of 6 dof manipulator by multirate control with depth identification. In *42nd IEEE Conference on Decision and Control (CDC)*, volume 5, 2003. (Cited on page 11.)

[50] Samsung Galaxy S21 FE 5G. `https://www.samsung.com/nl/smartphones/galaxy-s21-5g/galaxy-s21-fe-5g/`. [Online; accessed 04-Jan-2022]. (Cited on page 4.)

[51] Stefan Valentin Gheorghita, Martin Palkovic, Juan Hamers, Arnout Vandecappelle, Stelios Mamagkakis, Twan Basten, Lieven Eeckhout, Henk Corporaal, Francky Catthoor, Frederik Vandeputte, et al. System-scenario-based design of dynamic embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1), 2009. (Cited on pages 10, 25, and 27.)

[52] Graham Goodwin, Peter Ramadge, and Peter Caines. Discrete-time multivariable adaptive control. *IEEE Transactions on Automatic Control (TACON)*, 25(3), 1980. (Cited on page 124.)

[53] Dip Goswami, Alejandro Masrur, Reinhard Schneider, Chun Jason Xue, and Samarjit Chakraborty. Multirate controller design for resource-and schedule-constrained automotive ecus. In *Design, Automation & Test in Europe Conference (DATE)*, 2013. (Cited on page 10.)

[54] Volkan Gunes, Steffen Peter, Tony Givargis, and Frank Vahid. A survey on concepts, applications, and challenges in cyber-physical systems. *KSII Transactions on Internet and Information Systems (TIIS)*, 8(12):4242–4268, 2014. (Cited on page 1.)

[55] Andreas Hansson, Kees Goossens, Marco Bekooij, and Jos Huisken. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1), 2009. (Cited on pages 7, 26, and 77.)

[56] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. *Cambridge university press*, 2003. (Cited on page 46.)

[57] Soheil Hashemi, Hokchhay Tann, Francesco Buttafuoco, and Sherief Reda. Approximate computing for biometric security systems: A case study on iris scanning. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 319–324, March 2018. (Cited on pages 11, 139, and 140.)

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. (Cited on page 166.)

[59] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy. Learning Lightweight Lane Detection CNNs by Self Attention Distillation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1013–1021, Oct 2019. (Cited on page 137.)

[60] Seo-Hyun Jeon, Jin-Hee Cho, Yangjae Jung, Sachoun Park, and Tae-Man Han. Automotive hardware development according to iso 26262. In *13th international conference on advanced communication technology (ICACT)*, pages 588–592. IEEE, 2011. (Cited on page 5.)

[61] Jetson AGX Xavier: Deep Learning Inference Benchmarks. `https://developer.nvidia.com/embedded/jetson-agx-xavier-dl-inference-benchmarks`. [Online; accessed 29-Dec-2021]. (Cited on page 166.)

[62] Haomiao Jiang, Qiyuan Tian, Joyce Farrell, and Brian A Wandell. Learning the image processing pipeline. *IEEE Transactions on Image Processing*, 26(10):5032–5042, Oct 2017. (Cited on pages 11, 139, and 167.)

[63] Honglan Jiang, Francisco J. H. Santiago, Mohammad Saeed Ansari, Leibo Liu, Bruce F. Cockburn, Fabrizio Lombardi, and Jie Han. Characterizing approximate adders and multipliers optimized under different design constraints. In *Proceedings of the on Great Lakes Symposium on VLSI*, GLSVLSI '19, page 393–398, New York, NY, USA, 2019. Association for Computing Machinery. (Cited on pages 11 and 139.)

[64] Chaitanya Jugade, Daniel Hartgers, Phan Duc Anh, Sajid Mohamed, Mojtaba Haghi, Dip Goswami, Andrew Nelson, Gijs van der Veen, and Kees Goossens. An Evaluation Framework for Vision-in-the-Loop Motion Control Systems. In *Emerging Technologies and Factory Automation (ETFA)*, 2022. (Cited on page 207.)

[65] Akihiro Kawamura, Kenji Tahara, Ryo Kurazume, and Tsutomu Hasegawa. Robust visual servoing for object manipulation with large time-delays of visual information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. (Cited on page 11.)

[66] Srinidhi Kestur, Mi Sun Park, Jagdish Sabarad, Dharav Dantara, Vijaykrishnan Narayanan, Yang Chen, and Deepak Khosla. Emulating mammalian vision on reconfigurable hardware. In *IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012. (Cited on page 11.)

[67] Younghoon Kim, Swagath Venkataramani, Nitin Chandrachoodan, and Anand Raghunathan. Data subsetting: A data-centric approach to approximate computing. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 576–581, 2019. (Cited on page 139.)

[68] J Kosecka, R Blasi, CJ Taylor, and Jitendra Malik. Vision-based lateral control of vehicles. In *Proceedings of Conference on Intelligent Transportation Systems*, 1997. (Cited on pages 14, 20, 27, 57, and 179.)

[69] Peter Krautgartner and Markus Vincze. Performance evaluation of vision-based control tasks. In *Proceedings of International Conference on Robotics and Automation (ICRA)*, volume 3, 1998. (Cited on pages 11, 12, 14, 56, 69, 71, 74, 75, 114, and 115.)

[70] Marco Lattuada and Fabrizio Ferrandi. Modeling pipelined application with synchronous data flow graphs. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2013. (Cited on page 76.)

[71] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. (Cited on page 5.)

[72] Edward A Lee. Cyber physical systems: Design challenges. In *11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, pages 363–369. IEEE, 2008. (Cited on page 1.)

[73] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9), 1987. (Cited on pages 31 and 90.)

[74] Edward A Lee and Alberto Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 17(12), 1998. (Cited on page 10.)

[75] Kibeom Lee, Shengbo Eben Li, and Dongsuk Kum. Synthesis of robust lane keeping systems: Impact of controller and design parameters on system performance. *IEEE Transactions on Intelligent Transportation Systems*, 20(8):3129–3141, 2019. (Cited on page 162.)

[76] Sugie Lee, Hyunbin Moon, Yeri Choi, and Dong Keun Yoon. Analyzing thermal characteristics of urban streets using a thermal imaging camera: A case study on commercial streets in seoul, korea. *Sustainability*, 10(2):519, 2018. (Cited on page 4.)

[77] Shuai Li, Ce Zhu, Yanbo Gao, Yimin Zhou, Frédéric Dufaux, and Ming-Ting Sun. Lagrangian multiplier adaptation for rate-distortion optimization with inter-frame dependency. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 2015. (Cited on pages 11, 14, 56, 75, and 103.)

[78] You Li and Javier Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37(4):50–61, 2020. (Cited on page 4.)

[79] Bo Lincoln and Bo Bernhardsson. Optimal control over networks with long random delays. In *Proceedings of the International Symposium on Mathematical Theory of Networks and Systems*, volume 7, 2000. (Cited on pages 105 and 107.)

[80] Gareth A Lloyd and Steven J Sasson. Electronic still camera, December 26 1978. US Patent 4,131,919. (Cited on page 3.)

[81] Gigel Macesanu, Vasile Comnac, Florin Moldoveanu, and Sorin M Grigorescu. A time-delay control approach for a stereo vision based human-machine interaction system. *Springer Journal of Intelligent & Robotic Systems*, 76(2), 2014. (Cited on page 11.)

[82] Robinson Medina, Sander Stuijk, Dip Goswami, and Twan Basten. Implementation-aware design of image-based control with on-line measurable variable-delay. In *Design, Automation & Test in Europe Conference (DATE)*, 2019. (Cited on pages 11, 56, 69, 71, 114, and 115.)

[83] Alexandre Mercat, Justine Bonnot, Maxime Pelcat, Wassim Hamidouche, and Daniel Menard. Exploiting computation skip to reduce energy consumption by approximate computing, an hevc encoder case study. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 494–499, 2017. (Cited on pages 11 and 139.)

[84] Olivier Michel. Cyberbotics Ltd. Webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems (IJARS)*, 1(1):5, 2004. (Cited on pages 51, 116, and 141.)

[85] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954*, 2020. (Cited on page 167.)

[86] Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *58th Conference on Decision and Control (CDC)*, pages 3997–4004. IEEE, 2019. (Cited on pages 16, 74, 79, 87, 121, and 206.)

[87] Sajid Mohamed, Sayandip De, Konstantinos Bimpisidis, Vishak Nathan, Dip Goswami, Henk Corporaal, and Twan Basten. IMACS: A Framework for Performance Evaluation of Image Approximation in a Closed-loop System. In *8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2019. (Cited on pages 17, 116, 135, 140, and 206.)

[88] Sajid Mohamed, Dip Goswami, and Twan Basten. Bridging the controller design-implementation gap for image-based control systems. In *ICT.OPEN*, 2019. (Cited on page 207.)

[89] Sajid Mohamed, Dip Goswami, and Twan Basten. Matlab2Trace: A Matlab to Trace translator to visualise and analyse concurrent system activities and execution traces. In *8th MCAA Annual Conference*, 2021. (Cited on page 207.)

[90] Sajid Mohamed, Dip Goswami, Sayandip De, and Twan Basten. Optimising multiprocessor image-based control through pipelining and parallelism. *IEEE Access*, 9:112332–112358, 2021. (Cited on pages 15, 73, and 206.)

[91] Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario-and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 75:103037, 2020. (Cited on pages 13, 23, 75, 117, and 206.)

[92] Sajid Mohamed, Nilay Saraf, Daniele Bernardini, Dip Goswami, Twan Basten, and Alberto Bemporad. Adaptive predictive control for pipelined multiprocessor image-based control systems considering workload variations. In *59th IEEE Conference on Decision and Control (CDC)*, 2020. (Cited on pages 14, 55, and 206.)

[93] Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. Optimising quality-of-control for data-intensive multiprocessor image-based control systems considering workload variations. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 320–327, 2018. (Cited on pages 13, 23, 75, 123, and 207.)

[94] Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. DASA: An open-source design, analysis and simulation framework for automotive image-based control systems. In *6th MCAA Annual Conference*, 2019. (Cited on page 207.)

[95] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998. (Cited on pages 3 and 5.)

[96] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 286–291, 2018. (Cited on pages 166 and 167.)

[97] NVIDIA DRIVE end-to-end platform for software-defined autonomous vehicles. `https://www.nvidia.com/en-us/self-driving-cars/drive-platform/`. [Online; accessed 29-Dec-2021]. (Cited on pages 5, 8, 46, and 47.)

[98] NVIDIA LaneNet, High-Precision Lane Detection. `https://blogs.nvidia.com/blog/2019/07/10/drive-labs-neural-nets-predict-lane-lines`. [Online; accessed 29-Dec-2021]. (Cited on pages 116 and 141.)

[99] NVIDIA Nsight and Visual Studio Edition. 3.0 user guide. *NVIDIA Corporation*, 2013. (Cited on page 52.)

[100] Katsuhiko Ogata et al. Discrete-time control systems. *Prentice Hall Englewood Cliffs, NJ*, 2, 1995. (Cited on pages 56, 106, and 123.)

[101] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017. (Cited on pages 5, 24, 55, and 122.)

[102] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 519–530, 2013. (Cited on page 144.)

[103] Arnab Raha, Ankush Chakrabarty, Vijay Raghunathan, and Gregery T Buzzard. Embedding approximate nonlinear model predictive control at ultra-high speed and extremely low power. *IEEE Transactions on Control Systems Technology (TCST)*, pages 1–8, 2019. (Cited on page 146.)

[104] Arnab Raha and Vijay Raghunathan. Approximating beyond the processor: Exploring full-system energy-accuracy tradeoffs in a smart camera system. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 26(12):2884–2897, Dec 2018. (Cited on pages 11, 139, 140, and 146.)

[105] Arnab Raha, Soubhagya Sutar, Hrishikesh Jayakumar, and Vijay Raghunathan. Quality Configurable Approximate DRAM. *IEEE Transactions on Computers*, 66(7):1172–1187, 2017. (Cited on page 139.)

[106] Arnab Raha, Swagath Venkataramani, Vijay Raghunathan, and Anand Raghunathan. Energy-efficient reduce-and-rank using input-adaptive approximations. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 25(2):462–475, 2017. (Cited on page 140.)

[107] Randy Frank. Steering in the Right Direction. *Electronic Design*, 2016. (Cited on pages 116, 141, and 146.)

[108] Ashish Ranjan, Arnab Raha, Vijay Raghunathan, and Anand Raghunathan. Approximate memory compression. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(4):980–991, 2020. (Cited on page 139.)

[109] Felix Richter. Smartphones Wipe Out 40 Years of Camera Industry Growth. `https://www.statista.com/chart/15524/worldwide-camera-shipments/`, 2021. [Online; accessed 29-Dec-2021]. (Cited on pages 3 and 179.)

[110] Selma Saidi, Sebastian Steinhorst, Arne Hamann, Dirk Ziegenbein, and Marko Wolf. Future automotive systems design: research challenges and opportunities: special session. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2018. (Cited on pages 5, 10, 11, 16, 55, 56, 122, and 131.)

[111] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. Load value approximation. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 127–139, 2014. (Cited on page 139.)

[112] Róbinson Medina Sánchez, Juan Valencia, Sander Stuijk, Dip Goswami, and Twan Basten. Designing a controller with image-based pipelined sensing and additive uncertainties. *ACM Transactions on Cyber-Physical Systems (TCPS)*, 3(3), 2019. (Cited on pages 11, 14, 26, 41, 43, 44, 56, 61, 69, 71, 74, 75, 114, 115, and 181.)

[113] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming dr. frankenstein: Contract-based design for cyber-physical systems. *European Journal of Control*, 18(3), 2012. (Cited on page 10.)

[114] Teodora Sanislav and Liviu Miclea. Cyber-physical systems-concept, challenges and research areas. *Journal of Control Engineering and Applied Informatics*, 14(2):28–33, 2012. (Cited on page 1.)

[115] Nilay Saraf and Alberto Bemporad. An efficient non-condensed approach for linear and nonlinear model predictive control with bounded variables. in arXiv preprint arXiv:1908.07247, 2019. (Cited on pages 57, 68, 69, and 70.)

[116] Steven J Sasson and Robert G Hills. Electronic still camera utilizing image compression and digital storage, May 14 1991. US Patent 5,016,107. (Cited on page 3.)

[117] Paul M Sharkey and David W Murray. Delays versus performance of visually guided systems. *IEE Proceedings - Control Theory and Applications*, 143(5), 1996. (Cited on pages 7 and 74.)

[118] Savvas Sioutas, Sander Stuijk, Twan Basten, Henk Corporaal, and Lou Somers. Schedule synthesis for halide pipelines on GPUs. *ACM Transactions on Architecture and Code Optimization (TACO)*, 17(3):1–25, 2020. (Cited on page 150.)

[119] Firew Siyoum, Marc Geilen, and Henk Corporaal. Symbolic analysis of dataflow applications mapped onto shared heterogeneous resources. In *51st Annual Design Automation Conference (DAC)*, 2014. (Cited on page 34.)

[120] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 36(7), 2013. (Cited on pages 11, 14, 56, 75, and 103.)

[121] Sundararajan Sriram and Shuvra S Bhattacharyya. Embedded multiprocessors: Scheduling and synchronization. *CRC press*, 2018. (Cited on page 90.)

[122] Sander Stuijk. *Predictable mapping of streaming applications on multiprocessors*. PhD thesis, Eindhoven University of Technology, 2007. (Cited on pages 8, 18, 79, and 83.)

[123] Sander Stuijk, Marc Geilen, and Twan Basten. SDFˆ3: SDF for free. In *6th International Conference on Application of Concurrency to System Design (ACSD)*, 2006. (Cited on pages 35, 76, 84, 90, and 97.)

[124] Sander Stuijk, Marc Geilen, and Twan Basten. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *Euromicro Conference on Digital System Design (DSD)*, 2010. (Cited on page 10.)

[125] Zhendong Sun and Shuzhi Sam Ge. Stability theory of switched dynamical systems. *Springer Science & Business Media*, 2011. (Cited on pages 30, 44, 78, and 102.)

[126] Emil Talpes, Debjit Das Sarma, Ganesh Venkataramanan, Peter Bannon, Bill McGee, Benjamin Floering, Ankit Jalote, Christopher Hsiong, Sahil Arora, Atchyuth Gorti, et al. Compute solution for tesla's full self-driving computer. *IEEE Micro*, 40(2):25–35, 2020. (Cited on pages 5 and 8.)

[127] Camillo J Taylor, Jana Košecká, Robert Blasi, and Jitendra Malik. A comparative study of vision-based lateral control strategies for autonomous highway driving. *International Journal of Robotics Research*, 18(5), 1999. (Cited on page 123.)

[128] Tesla: Future of driving. `https://www.tesla.com/autopilot`. [Online; accessed 04-Jan-2022]. (Cited on page 4.)

[129] Bart D Theelen, Marc CW Geilen, Twan Basten, Jeroen PM Voeten, Stefan Valentin Gheorghita, and Sander Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Formal Methods and Models for Co-Design (MEMOCODE)*, 2006. (Cited on pages 10, 26, 31, 76, 81, and 125.)

[130] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, 2000. (Cited on page 10.)

[131] Ye Tian, Qian Zhang, Ting Wang, Feng Yuan, and Qiang Xu. Approxma: Approximate memory access for dynamic precision scaling. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, page 337–342, New York, NY, USA, 2015. Association for Computing Machinery. (Cited on pages 11 and 139.)

[132] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *International Conference on Software Engineering (ICSE)*. ACM, 2018. (Cited on pages 14 and 57.)

[133] Joe Triggs and Derek Burke. Camera link technology for automotive applications. *ATZelectronics worldwide*, 15(7):18–22, 2020. (Cited on pages 4 and 179.)

[134] The TuSimple Lane Detection Challange. `https://github.com/TuSimple/tusimple-benchmark`. [Online; accessed 29-Dec-2021]. (Cited on page 161.)

[135] EP van Horssen. *Data-intensive feedback control: switched systems analysis and design*. PhD thesis, Eindhoven University of Technology, 2018. (Cited on pages 5, 11, 24, 122, 123, and 124.)

[136] Yafei Wang, Binh Minh Nguyen, Hiroshi Fujimoto, and Yoichi Hori. Multirate estimation and control of body slip angle for electric vehicles based on onboard vision system. *IEEE Transactions on Industrial Electronics (TIE)*, 61(2), 2014. (Cited on page 11.)

[137] Ze Wang, Weiqiang Ren, and Qiang Qiu. LaneNet: Real-Time Lane Detection Networks for Autonomous Driving. *arXiv preprint arXiv:1807.01726*, 2018. (Cited on page 46.)

[138] Peter Welch. The use of FFT for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio & Electroacoustics*, 15(2):70–73, 1967. (Cited on page 126.)

[139] Nicky J Welton and AE Ades. Estimation of markov chain transition probabilities and rates from fully and partially observed data: uncertainty propagation, evidence synthesis, and model calibration. *Medical Decision Making*, 25(6):633–645, 2005. (Cited on pages 122 and 132.)

[140] Ming Yang, Nathan Otterness, Tanya Amert, Joshua Bakita, James H. Anderson, and F. Donelson Smith. Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. In *ECRTS*, pages 20:1–20:21, 2018. (Cited on page 48.)

[141] Imaging technologies are transforming the automotive industry. `http://www.yole.fr/ImagingForAuto_IndustryReview.aspx#.YdWbAlXMKYl`. [Online; accessed 04-Jan-2022]. (Cited on page 4.)

[142] Peter Colin Young and JC Willems. An approach to the linear multivariable servomechanism problem. *International Journal of Control*, 15(5):961–979, 1972. (Cited on page 42.)

[143] Majid Zamani, Soumyajit Dey, Sajid Mohamed, Pallab Dasgupta, and Manuel Mazo. Scheduling of controllers' update-rates for residual bandwidth utilization. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 85–101. Springer, 2016. (Cited on page 207.)

[144] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. Robust and optimal control. *Prentice hall New Jersey*, 40, 1996. (Cited on page 42.)

[145] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 547–560, 2018. (Cited on pages 11, 139, and 140.)

# Acknowledgements

*I take this opportunity to express my heartfelt gratitude to all the fantastic people who have come across my life and whose actions impacted me positively, knowingly or unknowingly.*

This PhD, for me, has been a positive life-changing roller-coaster ride full of ups and downs. The PhD journey reinvigorated my childhood inquisitiveness and critical thinking that had been mellowed down with my conditioned education. This journey showed me the fruitfulness of patience, perseverance and dedication, a journey that taught me directness, time management, and prioritisation and instilled in me confidence, courage and passion for innovation. All this would not have been possible without the help of numerous extraordinary individuals in my life. A heartfelt shout-out to all of them for accommodating me even when I have been socially awkward as an introvert.

First and foremost. I want to thank Allah (God Almighty) for showering his blessing upon me. I believe in God; for me, God is the supernatural force that drives our universe, and I am grateful to God for reaching this far. Alhamdulillah!

Next, I would like to thank my supervisor, dr. Dip Goswami, and my promotor, prof.dr.ir. Twan Basten, for giving me this exciting opportunity to pursue my PhD at TU/e as part of the oCPS project, which was a Marie Curie Horizon 2020 Innovative Training Network programme. The initial research proposal was to co-design control and streaming applications while optimising quality-of-control and quality-of-service. It was a tremendous challenge for me initially to understand the complexity of the problem, as my previous research experience was on formal verification. I sincerely thank you for your patience during this initial phase and for pointing me in the right direction. Thank you for believing in me and making me a better researcher and human being.

Dip, thank you for being an integral part of my PhD journey. You have given me freedom, valuable guidance and wholehearted support throughout my research. I appreciate your openness and help throughout my PhD. Thanks for patiently guiding me during my initial two years of struggle (which helped shape my research direction). I also enjoyed being a teaching assistant in your courses. Thank you for allowing me to hone my teaching skills and giving me the necessary freedom. Thank you for entrusting me with the mentorship of many master's theses and bachelor's projects. I grew a lot as a researcher, teacher and mentor with your guidance. I also thoroughly enjoyed our time together during the business trips we shared as part of the oCPS program and conferences. Thank you for recommending me to ITEC and guiding me in this career transition towards the industry.

Thanks a lot for also maintaining a personal relationship and inviting Jemshi and me to memorable occasions and festivals. We always had a wonderful time (and food :D), mostly at your home with you and your family (Trisita ma'am and Amy). I sincerely thank you for maintaining our research collaboration and personal interactions even after I started working in the industry.

Twan, thank you for making me a better researcher and showing me how to objectively assess research while maintaining a healthy personal relationship. You taught me meticulously and painstakingly to help improve my writing, presentation and research skills. I am awed by your perfectionism, which immensely improved the quality of the research output. A trivia of your perfectionism is that all my first author publications with you (including the journal papers) have been accepted almost as it is for the first time, with only minor revisions. Though the process was frustrating at times due to how long it took, the result has been worth it. Also, thank you for showing me the importance of empathy and being direct. I sincerely appreciate our open and direct discussions, especially during the bad phase of my research. Thank you for the offer to continue as University Researcher for another year at TU/e. Finally, I express my gratitude to you for hosting memorable ES days when you were the chair of the ES group. I am glad I could spend an incredible five years with you, and I hope we can continue our interaction.

I want to thank the members of my PhD defense committee – prof.dr.ir. Jeroen Voeten, prof.dr. Kees Goossens, prof.dr. Axel Jantsch and prof.dr.sc. Samarjit Chakraborty - for their time and effort in reviewing my thesis and for giving me valuable feedback and pointers for improvement of this thesis.

My PhD research was part of the oCPS program, and I want to thank my friends and colleagues who were part of the oCPS consortium sincerely – Sayandip de, Amr Ibrahim, Deepak Vedha Raj Sudhakar, Hadi Balaghiinaloo, Asad Ullah Awan, Nilay Saraf, David Juhasz, Marina Rantanen Modeer, Pouya Mahdavipour, Precious Ugo Abara, Mladen Cicic, Tahira Iqbal, Aida Rashidinejad, Giovanni Ranuzzi, Myrthe van Delft - for all the collaborations, interactions, courses and trips we shared.

Sayan, it was an immense pleasure to collaborate with you. You are brilliant, talented, hardworking and humble. What started as a simple idea that we had at the coffee corner (regarding the evaluation of approximation in an image-based control loop) has resulted in one journal and three conference publications (along with a best-paper award). I have learned a lot from you, and I already miss our collaboration. Sincere thanks for also being a fantastic friend. Your friendship has made my PhD journey sweet and memorable professionally and personally.

Amr, JazakAllahu Khairan. Your friendship helped me overcome most of the difficult times during my thesis. I could discuss with you any topic that was bothering me. You always had a positive attitude and a smiling face, even though my problems seemed minuscule compared to all your difficulties. You inspired me as a Muslim and have guided me with your knowledge and actions. Thank you!

Asad, your research acumen, knowledge and actions have inspired me immensely. I enjoyed collaborating with you, and thanks for helping me get my first top-tier control publication (in CDC). Nilay, thanks for helping me with the second

CDC publication and for your friendship. Thanks for introducing me to the world of real-time MPC. Deepak, thanks for being a helpful host and friend during my secondment at Inchron (Potsdam, Berlin). Hadi, your cheerfulness is contagious. You are an inspiration to me for showing me that we can overcome adversity with sheer determination.

I also want to thank my co-authors and other research collaborators who have (in)directly contributed to this dissertation - prof. Henk Corporaal, prof. Alberto Bemporad, dr. Daniele Bernardini, dr. Marc Geilen, and MSc students (now graduates!) – Diqing Zhu, Konstantinos Bimpisidis (Kostas), Vishak Nathan, Yingkai Huang, Varun Shankar, Chaolun Ma, Rahul Ramesh, and Mingyu An.

Henk, I hope I can be at least half as cheerful, energetic, enthusiastic and knowledgeable as you are when I reach your age. Your constant drive to improve yourself, curiosity and efficiency have profoundly impacted me.

Marc, you are the most humble, knowledgeable, hands-on researcher and professor I have ever met. I could drop in at any time at your office with my (often stupid) questions regarding data flow. You always made me think and improve as a researcher. I also enjoyed all the trips we had as part of tutorials/conferences and our personal interactions. I consider it my blessing to know you.

Kostas, your technical skills are astounding. Thanks for giving us the solid foundation to continue the work on the approximation-aware design. Diqing, thanks for your initial work on the DASA framework, which was my thesis's starting point. Vishak, it was a pleasure to have worked with you. Being from my hometown, it was a respite to talk to you in Malayalam during our (un)official meetings.

The Electronics Systems (ES) group hosted my PhD at the Eindhoven University of Technology. Here, I had the privilege to work alongside some intelligent, talented and kind colleagues. Alessandro Frigerio, Ali Banagozar, Alireza Mohammadkhani, Amir Behrouzian, Andrew Nelson, Barry de Bruin, Berk Ulker, Bram van der Sanden, Chaitanya Jugade, Cumhur Erdin, Emad Ibrahim, Floran de Putter, Gabriela Breaban, Gagandeep Singh, Hadi Ara, Hadi Balef, Hailong Jiao, Hamideh Hajizadeh, Hossein Elahi, Ilde Lorato, Joost van Pinxten, João Bastos, Jos Huisken, Juan Valencia, Kamlesh Singh, Kanishkan Vadivel, Lech Jozwiak, Lorenzo Chelini, Luc Waeijen, Mahsa Mousavi, Majid Nabi, Manil Dev Gomony, Martijn Koedam, Martin Roa Villescas, Mark Wijtvliet, Mladen Skelin, Mohammad Emad, Mohammad Tahghighi, Mojtaba Haghi, Paul Detterer, Rasool Tavakoli, Reinier van Kampenhout, Robinson Medina Sanchez, Roel Jordans, Roohallah Azarmi, Ruben Jonk, Saeid Dehnavi, Savvas Sioutas, Shayan Tabatabaei Nikkhah, Shima Sedighiani, Sherif Eissa, Siamack Beigmohammadi, Stefano Corda, Syed Rehan Afzal, Umar Waqas, Vibhor Jain, Victor Sanchez, Wenjin Wang, Yonghui Li and Zhan Gao, thank you all for a memorable time and enjoyable ES days. A special thanks to Sander Stuijk and Twan for organising the ES days. I am also grateful to the ES group's secretary Marja de Mol for her kindness.

I would also like to thank my current employer, ITEC B.V., for supporting me and providing me the freedom to continue collaborating with TU/e. Sincere gratitude to Joep Stokkermans (innovation director) and colleagues - Gijs van

der Veen, Leonard Lensink, Jan Driessen, Raymond Rosmalen, Jasper Wesselingh, Hans Kuipers, Raymond Krabbenborg, Erik Stens, Anass Fakir and many more.

I want to thank the teachers who made me who I am today. Thanks, Prof. Partha Pratim Chakrabarti, Samarjit, and Prof. Pallab Dasgupta, for guiding and instilling a passion for research in me. Sincere heartfelt thanks to my school teachers – Nimmy teacher (Nirmala Joseph), Anuradha teacher, Francis sir, Sambhu sir, Rajendran sir and Sharadha teacher.

The contribution of friends to keeping my sanity during the tumultuous PhD journey needs special mention. Kanishkan ji, thanks a lot for providing selfless help throughout. I do not know what I would have done without your support in the last few years. I feel blessed to have you as my friend. Thanks also to Akshaya for being a kind host. Ruben, you are the first Dutch friend I had and thanks for helping me with all the Dutch letters and bureaucracy. Thanks for introducing me to board games and getting me hooked on them. I really enjoyed our times together and thanks also to Esther. Alessandro, thanks for organising all the informal social events and bringing in a warm feeling within the ES group (though I could be part of it only for the first year, I had a memorable time). Thanks, Ruben, Paul, Cumhur, Alessandro, Kanishkan, and Sayan, for all the board game evenings we shared. I also thank the members of the ES football group for all the fun futsal games we played. Thanks also to the 'Kombans football group' and 'Eindhoven Malayalees' for making our stay at Eindhoven fun-filled. Special thanks to Bejoy and Prajitha for keeping this active. Heartfelt thanks also to the Salaam Student Association for the warm welcome, events and celebrations during the festivities.

Sincere thanks to friends in Vreden - Ha Dang, Raisul, Tofa, Gowri *akka* & *appa*, and others - for keeping us engaged with trips, food & games during the last years.

Thanks a lot also to the close friends who are from Kerala and were part of this journey. Sajith Muhammed, the wavelength I shared with you is unique. I do not think anyone else understands and resonates with me better than you right now. Thanks for all the help and pointers and for sharing the latest tricks to financial management. Jemshi and I also enjoyed our time with you, Aysha and Aymu. Nazneen Mansoor and Raslam Showkath, I cannot thank you enough for being there for Jemshi while she was in Berlin. Thanks a ton for the food, the games and the fun we have had at your place. Haneef Shijaz and Thasni, you have been the best neighbours Jemshi and I could have hoped for in Eindhoven. The time we have spent together has been a much-needed relief from the work-related topics and helped us to reminisce about our life back in Kerala. Raees, thanks for letting me crash at your place on short notice. Sarath Menon, thanks for being a kind host! Sincere gratitude to – Rizwin Shooja, Rifat Kamarudheen, Rahul Sivakumar, Arun B N, Arjun, Anoop Kodakkal, Joby Joseph, Shehna, Pallavi and many others who made life in Europe colourful. Special mention to the 'SV Football' group – Vishnu (odiyan), Xavier (janu), Deepak (p2), Sunish, Jonny, B N, Subhro, Tharun, Karthik (Baba), Gautham M D, Harif, Bibin, Damien - whose constant rants targeting Chelsea and their support for rival clubs have given me a podium to vent my feelings and have everlasting discussions about almost anything. Thanks a lot

to other Sarvodaya Vidyalaya (SV) friends also for keeping in touch and giving me something to look forward to when visiting Trivandrum – Ram Krishnan, Tharaj Thaj, Sidharth Radhakrishnan, Finny, Feros, Vishnu (rvp), Aarif, Aswin, Vivek, Nirmal, Rajiv, Vishnu V Gopal and many more. Special thanks to the 'danger boys' – Sunit Mathew, Rohit (panku), Mihir Pillai, Nishant Roy, Avinesh Vasudevan, Arkii, Shankar and Mevin – for the 'among us' games, fantasy premier league and the unique sense of humour. Thanks a lot also to my close KGP friends - Sanu Ann Abraham, Dhanesh Chandran, Vignesh, Jipin, Muhammed (Ammachandy), Daleef Rahman, Sarath, Jithinlal, Akash Koppa, Shan and many more for keeping the friendship alive and for all the support.

I have been fortunate to have been part of a joint family in Trivandrum. I sincerely thank the 'Jamamina family' for the joy, happiness and comfort they have provided me over the years. Thanks a lot to *ummoomma* (Amina Umma), *mama* (Anwar Meeran), *chinnamma* (Anisha Banu), *mami* (Rahumath Rameesa), *atha* (Nazer), Farsana, Ashiq, Alameen, Aysha, Basil, Aslam, Ansar, Harith, Rana, Rania and Ahyaan. Sincere thanks to all the close relatives – *office uppa* (Shahul Hameed) & *ummoomma*, Sulfi *mama*, Nadeera *mami*, Nowfal, Thoufeeq, Sabeer *mama*, Sabeena *kochumma*, mother-in-law (Fathima), Nazish and many others.

I also take this opportunity to remember *uppa* (grandfather) Jamaludheen, who passed away during this journey (Inna lillahi va inna ilaihi raajioon). I recall hearing about your demise and my mind getting shut off, unable to think and act. I could not even see you one last time since your wish was to be buried as soon as possible. Thanks for your support, encouragement and love throughout my life.

The three persons I would like to thank the most for being the light in my life and who always uplifted me when I was in my deepest of lows are my ever-loving *umma* (mom) Fathima, my ever-caring *vappa* (dad) Maheen, and my beloved wife Jemshi (with whom I have spent the major part of the last decade - deeply in love and headstrong in arguments). This PhD has been the most arduous journey I have undertaken in my life, and without the love, care, support, motivation and trust of these three gems in my life, I would not have made it. I love you all (from the bottom of my heart).

Jemshi, we married in your final year of medicine studies in West Bengal. You travelled across continents to move in with me and decided to pursue your postgraduation in Germany (which you had never thought of until then), having a totally different healthcare system in arguably one of the most challenging languages to learn. So me doing a PhD with a rollercoaster of emotions did not help. You persevered with your determination and even cared for me during my deepest lows. You are a superwoman, and I am in awe. You ensured that I did not work during out-of-office hours (keeping my work-life balance), rested, exercised regularly, ate healthily, and enjoyed life. Thank you, and as this journey comes to a close, I pray for many more joyful years ahead together! In Shaa Allah!

Sajid Mohamed
Netherlands, November 2022

# List of Publications

## Thesis related

### Journal papers

1. Sajid Mohamed, Dip Goswami, Sayandip De, and Twan Basten. Optimising multiprocessor image-based control through pipelining and parallelism. *IEEE Access*, 9:112332–112358, 2021

2. Sayandip De, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Approximation-aware design of an image-based control system. *IEEE Access*, 8: 174568–174586, 2020

3. Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, and Twan Basten. A scenario-and platform-aware design flow for image-based control systems. *Microprocessors and Microsystems*, 75:103037, 2020

### Conference papers

1. Sajid Mohamed, Nilay Saraf, Daniele Bernardini, Dip Goswami, Twan Basten, and Alberto Bemporad. Adaptive predictive control for pipelined multiprocessor image-based control systems considering workload variations. In *59th IEEE Conference on Decision and Control (CDC)*, 2020

2. Sayandip De, Sajid Mohamed, Konstantinos Bimpisidis, Dip Goswami, Twan Basten, and Henk Corporaal. Approximation trade offs in an image-based control system. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1680–1685. IEEE, 2020

3. Sajid Mohamed, Asad Ullah Awan, Dip Goswami, and Twan Basten. Designing image-based control systems considering workload variations. In *58th Conference on Decision and Control (CDC)*, pages 3997–4004. IEEE, 2019

4. Sajid Mohamed, Sayandip De, Konstantinos Bimpisidis, Vishak Nathan, Dip Goswami, Henk Corporaal, and Twan Basten. IMACS: A Framework for Performance Evaluation of Image Approximation in a Closed-loop System. In *8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2019 *(Best paper award)*

5. Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. Optimising quality-of-control for data-intensive multiprocessor image-based control systems considering workload variations. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 320–327, 2018

## Others

### Journal paper

1. Majid Zamani, Soumyajit Dey, Sajid Mohamed, Pallab Dasgupta, and Manuel Mazo. Scheduling of controllers' update-rates for residual bandwidth utilization. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 85–101. Springer, 2016

### Conference papers

1. Chaitanya Jugade, Daniel Hartgers, Phan Duc Anh, Sajid Mohamed, Mojtaba Haghi, Dip Goswami, Andrew Nelson, Gijs van der Veen, and Kees Goossens. An Evaluation Framework for Vision-in-the-Loop Motion Control Systems. In *Emerging Technologies and Factory Automation (ETFA)*, 2022

2. Sayandip De, Yingkai Huang, Sajid Mohamed, Dip Goswami, and Henk Corporaal. Hardware- and situation-aware sensing for robust closed-loop control systems. In *Design, Automation and Test in Europe Conference (DATE)*, 2021

3. Martin Becker, Sajid Mohamed, Karsten Albers, Partha Pratim Chakrabarti, Samarjit Chakraborty, Pallab Dasgupta, Soumyajit Dey, and Ravindra Metta. Timing analysis of safety-critical automotive software: The AUTOSAFE tool flow. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 385–392. IEEE, 2015

### (Extended) Abstracts

1. Sajid Mohamed, Dip Goswami, and Twan Basten. Matlab2Trace: A Matlab to Trace translator to visualise and analyse concurrent system activities and execution traces. In *8th MCAA Annual Conference*, 2021

2. Sajid Mohamed, Dip Goswami, and Twan Basten. Bridging the controller design-implementation gap for image-based control systems. In *ICT.OPEN*, 2019

3. Sajid Mohamed, Diqing Zhu, Dip Goswami, and Twan Basten. DASA: An open-source design, analysis and simulation framework for automotive image-based control systems. In *6th MCAA Annual Conference*, 2019

# About the Author

**Sajid Mohamed** was born on July 6, 1990 in Trivandrum, Kerala, India. He studied at the National Institute of Technology (NIT) Calicut, where he obtained the Bachelor of Technology (B.Tech.) degree in Electrical & Electronics Engineering in 2012. He obtained his Master of Technology (M.Tech.) degree in Embedded Controls & Software in 2014 from the Indian Institute of Technology (IIT) Kharagpur. He was awarded the German Academic Exchange Service (DAAD) scholarship to pursue his Master's thesis on Timed Abstractions and Analysis of Distributed Real-Time Control Architectures at the Technical University of Munich (TUM) in the Institute for Real-Time Computer Systems. He was a research assistant at IIT Kharagpur from 2014 to 2016.

In 2016, Sajid was awarded the Marie Skłodowska-Curie Scholarship and joined the Electronic Systems group of Eindhoven University of Technology (TU/e) as an Early-Stage Researcher in the *Platform-aware Model-driven Optimization of Cyber-Physical Systems (oCPS)* project. The project focused on training a generation of young researchers in cross-disciplinary thinking and delivering industrially validated toolchains by bringing together the state of the practice through six key industrial players and the state-of-the-art through four top universities and one research institute across Europe. In 2020, Sajid continued working in TU/e as a University Researcher in the FitOptiVis project. The results of his research have led, among others, to several peer-reviewed publications and the contents of this dissertation. In July 2021, he started his current job as a Principal Software Engineer in the Innovation Team of ITEC B.V.