Optimising Quality-of-Control for Data-intensive Multiprocessor Image-Based Control Systems considering Workload Variations

Sajid Mohamed[1], Diqing Zhu[1], Dip Goswami[1], Twan Basten[1,2]
[1]Electronic Systems Group, Eindhoven University of Technology, The Netherlands
[2]ESI, TNO, Eindhoven, The Netherlands
{s.mohamed, d.zhu, d.goswami, a.a.basten}@tue.nl

***Abstract*—Image-Based Control (IBC) systems have a long sample period. Sensing in these systems consists of compute-intensive image processing algorithms whose response times are dependent on image workload. IBC systems are typically designed for the worst-case workload that results in a long sample period and hence suboptimal quality-of-control (QoC). This worst-case based design is further considered for mapping of controller tasks and allocating platform resources, resulting in significant resource over-provisioning. Our design philosophy is to sample as fast as possible to optimise QoC for a given platform allocation, and for this, we present a structured design flow. Workload variations determine how fast we can sample and we model this dynamic behaviour using the concept of workload scenarios. Our choice of scenario-aware dataflow as the formal model for our application 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 sample period, and thus combine dataflow analysis and mapping with control design parameters and QoC to identify system scenarios, and iii) to efficiently implement a run-time mechanism that manages necessary dynamic reconfiguration between system scenarios. Our results show that our design approach outperforms the worst-case based design with respect to optimising QoC and maximising effective resource utilisation.**

I. Introduction

Image-Based Control (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 developments in cyber-physical systems (CPS) [1]. IBC have become popular with the advent of efficient image processing algorithms and low-cost CMOS cameras with high resolution [2]. The combination of the camera and 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 [2–4], autonomous vehicles [5, 6], advanced driver assistance systems (ADAS) [7], electron microscopes [8] and visual navigation [9].

The state-of-the-art design for IBC systems does not consider workload variations in the processing of the incoming image stream. They are designed for the worst-case response time (WCRT) of all tasks [1]. To design a controller, we consider sequential execution of the sensing task, control compute task and actuating task. Execution time for the sensing task is dependent on image workload variations. The sampling period is the time between the start of two consecutive sensing tasks. A typical control design assumes that we always have worst-case image workload. This further leads to a long sampling period. Fig. 1 illustrates the worst-case scenario where the camera input frame rate is 30 frames per second (fps) and the sampling period with worst-case image workload (h_{wc}) is 100 ms. Frame rate determines the number of image frames that arrive per time unit. Due to a long sampling period, we see that multiple image frames will arrive before the next T_s task can execute. In Fig. 1, we see that 3 image frames arrive within one sampling period. Thus, a long sampling period results in degraded control performance and conservative design solutions [1]. Control performance is generally quantified using a QoC metric such as settling time.

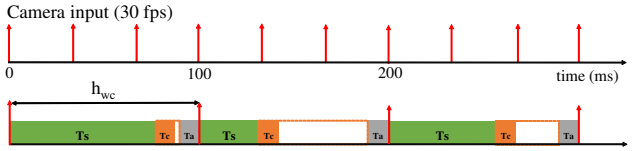

Fig. 1: Worst-case based design. T_s - sensing task, T_c - compute task, T_a - actuating task, h_{wc} - the sampling period with worst-case image workload

In a real world situation, the worst-case image workload is rarely encountered and the response time distribution due to workload variations can be statistically analysed (e.g. as a PERT distribution) [10]. Such a distribution helps the designer to classify frequently occurring *workload scenarios* and almost all the times we see that these frequent scenarios do not include the worst case. Intuitively, control performance can be improved if we take this into account. We can also avoid inefficient resource utilisation due to idling in a worst-case design, as shown in Fig. 1.

Our main contribution is a structured (co-)design flow for IBC systems to optimise QoC by sampling as fast as possible for a given platform allocation. Further, our method enables us to relate throughput, a parameter relevant for embedded systems engineers, to sample period, relevant for control engineer. This helps to design, analyse and optimise the IBC systems jointly from both embedded and control perspective.

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00063

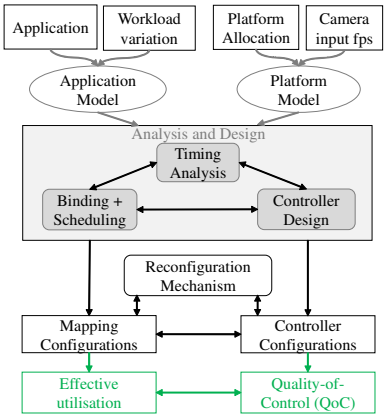

Fig. 2: Approach: Scenario- and Platform-Aware Design (SPADe) for IBC

The key idea is to characterize and model workload variations to identify a set of workload scenarios. For a given platform allocation, for each scenario, we find the optimal mapping configuration that maximises the effective resource utilization. Each workload scenario leads to a sampling period of the IBC system. A controller is designed for each sampling period optimising the QoC and ensuring switching stability of the overall system. The combination of a sampling period and the corresponding controller defines a controller configuration. The combination of controller and mapping configuration is a *system configuration*. At runtime, the IBC system switches (using a reconfiguration mechanism) between the system configurations depending on the actual image workload. As opposed to operating under the worst-case design with sampling period h_{wc}, the IBC system most of the time runs in system scenarios [11] with a shorter sampling period. A system scenario abstracts multiple workload scenarios having the same sampling period either due to platform constraints or due to our design choice. This reduces the average sampling period for the IBC system and improves its QoC.

Our Scenario and Platform-Aware Design (SPADe) flow is illustrated in Fig. 2.

SPADe involves the following design aspects:

1) Formal Modelling: i) identify and model the parameters that characterise workload variations, and ii) model application considering workload variations and platform considering platform constraints.
2) Analysis and Design: Analyse application and platform models to design system configurations.
3) Reconfiguration mechanism for run-time implementation.

This paper is organised as follows: related work is briefly discussed in Section II. We explain our problem setting in Section III. Background and preliminaries required to understand SPADe are detailed in Section IV. SPADe approach is explained in Sections V and VI. Section V explains formal modelling in SPADe and Section VI explains the analysis, design and implementation. Section VII presents the results of validation of our approach. Conclusion and future work are summarised in Section VIII.

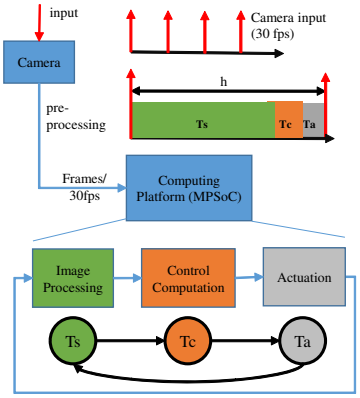

Fig. 3: Problem Setting

II. Related Work

Our SPADe approach models and explicitly considers the image processing in the sensing algorithm. One main challenge of IBC is the delay of the sensor data to reach the controller [2][9]. Generally, the analysis of the image processing step is limited to a measurement or estimate of the computational delay and is assumed to be negligible compared to other delays in the control loop [2][3][20]. Control engineers tackle a long sample period of IBC systems using state estimation [12], robust design [13], predictive control [14], observer-based [1], multi-rate sampling [26] and reconfigurable pipelining methods [15]. However, these approaches do not model and consider platform constraints like resource availability and mapping, and/or workload variations in image processing [1]. SPADe can explicitly handle these constraints using the proposed co-design approach. In [11], a system-scenario-based design approach is introduced from an embedded systems engineering perspective. However, control performance and long sample period are not explicitly considered. The SPADe approach bridges the design gap between a control engineer and an embedded systems engineer by relating image processing workloads and platform mappings to sample periods and quality of control.

III. Problem Setting

We consider a setting for an IBC system as shown in Fig. 3. A camera module captures the image stream and does preprocessing. This image stream is then fed to a multiprocessor platform, e.g a multiprocessor system-on-chip (MPSoC), at a fixed frame rate per second (fps), e.g. 30 fps. The control tasks run on this MPSoC. In this section, we present our problem setting.

A. Application Model

Our application is modelled using a model-of-computation (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 workload variations that happen at run-time. We assume that the worst-case execution times (WCET) of task workloads are given for a platform or can be computed, and the information on

parameters that relate to workload is known early on during the frame processing, e.g. from the frame header.

Our application is abstracted as sensing (T_s), control compute (T_c) and actuating (T_a) tasks. The execution sequence of the tasks is always $T_s \rightarrow T_c \rightarrow T_a$. The sampling period is then the time between the start of two consecutive sensing tasks. We define worst- and best-case sampling periods as sampling periods computed for maximal and minimal workloads. The best-case sampling period takes into account the WCET of the minimal (best-case) workload.

B. Platform Model

Our application runs on a multiprocessor platform with a tile-based architecture [19]. Each tile has a processor, a memory, a communication assist and a network interface. Tasks mapped onto the platform execute in a data-driven fashion for T_s and T_c, and in a time-triggered fashion for T_a, (e.g. as shown in Fig. 1). We assume that the WCET of a task can be bounded, i.e. the platform is predictable. A scheduler can do (re)configuration of the platform and time-triggered task execution. A mapping configuration determines the binding of application to the platform and its execution schedule. A controller configuration determines the time-triggered execution of T_a and the choice of controller gain for T_c. A system configuration refers to a combination of a mapping and a controller configuration. A reconfiguration mechanism realises runtime switching between system configurations with an overhead.

C. Platform Allocation

A platform allocation determines the resources that are allocated for a task or for an application. Resources that are allocated include: i) number of processors (could also be a part of a processor, e.g. slots in a time-division multiplexing (TDM) frame); ii) memory size - for local memory in a tile and/or shared memory, e.g. DDRAM; and iii) communication bandwidth. An application can have multiple binding options for a given platform allocation.

D. Design Questions

For a given application and a platform allocation, design

1) mapping configurations (bindings of application on the platform and execution schedules),
2) controller configurations (sampling periods and corresponding controller gains), and
3) a run-time reconfiguration mechanism,

such that we

- optimise QoC and
- maximise effective resource utilisation.

E. Running Example

We explain our SPADe approach using an example of a vision-based lateral control of a vehicle [20] where we want our vehicle to follow a lane autonomously (lane-keeping). A camera installed on the vehicle is our sensor. The camera sensor captures and pre-processes image frames at 30 fps.

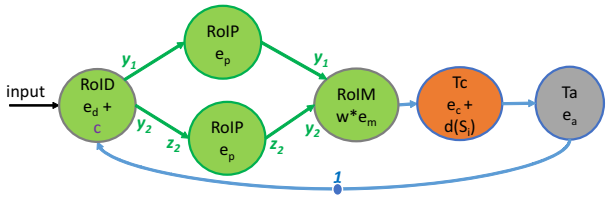

Fig. 4: Application model for platform with 2 processor tiles

An image processing algorithm then processes the frames and computes the lateral deviation (y_L) at a set look-ahead distance. Our controller takes y_L as the sensor input, computes the steering angle and actuates the steering to follow the lane.

IV. Background and Preliminaries

A. Synchronous DataFlow (SDF) Graphs

An SDF [22] is a directed graph $(\mathcal{A},\ \mathcal{C})$. A node $a \in \mathcal{A}$, called *actor*, models a task of the application and a platform-dependent time required for its execution. An edge $c \in \mathcal{C}$, called *channel*, captures (data) dependencies between actors. *Tokens* represent data communicated through channels. Channels may contain initial tokens, depicted with a solid dot. Initial tokens represent the availability of data in a channel at the start of analysis. Tokens on each channel also have a memory requirement.

An example SDF graph with 6 actors, 7 channels and 1 initial token is illustrated in Fig. 4 and is explained in Section V-B. A simulation of an SDF graph involves *firing* (execution) of actors. An actor executes for a fixed time specified by its execution time, e.g. in Fig. 4, e_p is the WCET for actor $RoIP$ (Region-of-Interest Processing). An actor can fire iff sufficient tokens are available on all its input channels. These amounts are called the *rates*, indicated next to the channel ends (omitted if 1). For an SDF, rates ($y_1,\ y_2,\ z_2$ in Fig. 4) should be a constant (e.g. $y_1 = y_2 = z_2 = 1$).

An essential property of SDFs is that every time an actor *fires*, it consumes the same number of tokens from its input channels and produces the same number of tokens on its output channels, as modelled by the rates. Fixed rates allow iterative execution of an SDF, where the initial token distribution occurs after each *iteration*. The number of actor firings required for one iteration is captured in the *repetition vector* [23]. Consistency (i.e., existence of a repetition vector) and absence of deadlock are practically, necessary conditions for analysis of SDFs which can be verified efficiently [22, 23].

Throughput is an important design constraint for streaming applications such as image processing [19]. Throughput of an SDF refers to how often an actor produces an output token. For our analysis, we use the throughput computation method from [25].

B. Scenario-Aware DataFlow (SADF) Graphs

An SADF graph [24] models the dynamic behaviour of an application using the concept of scenarios. A scenario represents an abstraction of a set of run-time behaviours. Each scenario is modelled using an SDF graph. Fig. 4 is an example of an SADF graph where scenarios use the same

SDF graph structure with different actor execution times and channel rates. The initial tokens in the SDF graphs capture the dependencies between subsequent scenarios. In our example (Fig. 4), RoID (Regions-of-Interest Detection) detects the workload, w. Scenarios are defined based on w. Scenario S_{bc} for minimal workload $w = w_{min} = 1$, has rates $y_1 = 1$, $y_2 = z_2 = 0$ and execution time of actor RoIM (Regions-of-Interest Merging) $= e_m$. Scenario S_{wc} for maximal workload $w = w_{max} = 6$, has rates $y_1 = 3$, $y_2 = 3$, $z_2 = 1$ and execution time of actor RoIM $= 6 * e_m$. The model captures the division of best-case and worst-case workload over two parallel processor tiles. More workload scenarios can be added, as explained in Section V-A.

C. Control Design

Image-based control (IBC) is a control application that is responsible for controlling a continuous-time plant defined by:

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t) \end{aligned} \tag{1}$$

where $x(t)$ is the *state* of the plant, $y(t)$ is the output of the plant, and $u(t)$ is the control input that is computed by a control algorithm or controller. A_c, B_c, and C_c are the state, input, and output matrices, respectively. These matrices model the behaviour of the system dynamics. For our running example, we consider the bicycle vehicle model in [20], where

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

where v_x: longitudinal velocity; v_y: lateral velocity; δ_f: front wheel steering angle; $\dot{\psi}$: the yaw rate of vehicle; l_f, l_r: distance of the front and rear axles from center of gravity (CoG); I_ψ: the total inertia of the vehicle around CoG; c_f, c_r: cornering stiffness of the front and rear tires; m: the total mass of the vehicle; L: the look-ahead distance of the camera; y_L: the lateral deviation from the desired centerline point at look-ahead distance; K_L: the curvature of the road at look-ahead distance; ε_L: the angle between the tangent to the road and the vehicle orientation;

The state vector $x(t) = [v_y, \dot{\psi}, y_L, \varepsilon_L, K_L]^T$, $y(t)$ is y_L, $u(t)$ is δ_f and $C_c = [0\ 0\ 1\ 0\ 0]$.

IBC is a feedback control loop and the control objective is to design $u(t)$ such that $y(t) \to 0$ (or a constant reference, r) with time (*regulation problem*). The time it takes for the system output $y(t)$ to reach and stay in a closed region (e.g., $< 2\%$) around the reference value is the *settling time* of IBC. We define QoC (denoted by Φ) as inverse of the settling time,

$$\Phi = \frac{1}{\text{settling time}} \tag{2}$$

Thus, a higher QoC Φ implies a shorter settling time and a better control performance.

1) Platform implementation: Implementation of IBC involves three sequential tasks: *sensing* (T_s), *computing* (T_c) and *actuating* (T_a). These tasks repeat; let the start and finish times of the k-th instance be given by $t_s(.)$ and $t_f(.)$, respectively. The execution times of T_s^k, T_c^k and T_a^k (the k-th instance) are given by,

$$e_t^k = t_f(T_t^k) - t_s(T_t^k) \tag{3}$$

where $t \in \{s, c, a\}$. The interval between two consecutive executions of sensing tasks T_s^k and T_s^{k+1} is then the *sampling period* h_k:

$$h_k = t_s(T_s^{k+1}) - t_s(T_s^k) \tag{4}$$

and $h_k \in H$, where H is the set of realisable sampling periods. Within each sampling period h_k, the control operations are executed sequentially (i.e., $T_s^k \to T_c^k \to T_a^k$). In addition, the time interval between the starting time of T_s^k and finishing time of T_a^k is known as the *sensor-to-actuator delay* τ_k,

$$\tau_k = t_f(T_a^k) - t_s(T_s^k). \tag{5}$$

2) Control task model: We consider a time-triggered implementation for T_a to guarantee constant τ_k. τ_k determines the time to trigger task T_a. A data-driven implementation approach is considered for T_s and T_c. Here, the sampling period can vary depending on the workload variations, resource availability and response times of individual tasks in different scenarios. This sampling period variation results in a switched system that could potentially destabilise the overall closed-loop system [16]. The designer needs to take this into account. For the data-driven implementation approach, the control input $u(t)$ is held (by a zero-order hold) until its next update. Using the model presented in [17], we have

$$x[k+1] = Ax[k] + B_0(\tau_k)u[k] + B_1(\tau_k)u[k-1] \tag{6}$$

where

$$\begin{aligned} A &= e^{A_c h_k},\ h_k \in H \\ B_0(h_k) &= \int_0^{h_k-\tau_k} e^{A_c s} ds \cdot B_c,\ B_1(h_k) = \int_{h_k-\tau_k}^{h_k} e^{A_c s} ds \cdot B_c \end{aligned}$$

In Eq. (6), 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 $z[0] = \begin{bmatrix} x[0] & 0 \end{bmatrix}^T$ obtaining the augmented higher order system

$$z[k+1] = A_{aug}(h_k)z[k] + B_{aug}(h_k)u[k] \tag{7}$$

where

$$A_{aug}(h_k) = \begin{bmatrix} A & B_1(h_k) \\ 0 & 0 \end{bmatrix},\ B_{aug}(h_k) = \begin{bmatrix} B_0(h_k) \\ I \end{bmatrix}$$

and I is the identity matrix. A check for controllability [27] is done for this augmented system. If the system is not controllable, controllability decomposition is done to obtain a controllable subsystem [28].

3) Control law: The control input $u[k]$ is a *state feedback* controller of the following form,

$$u[k] = K_k z[k] \tag{8}$$

where K_k is the state feedback gain at the k-th sample having period h_k. The design of K_k can be done with existing design

methods such as Linear Quadratic Regulator (LQR) or a pole-placement technique [27]. We define $K = K_1, \ldots, K_n$, such that K_i is the feedback gain for sampling period h_i, $h_i \in H$. The closed-loop system using Eq. (7) is given by,

$$z[k+1] = (A_{aug}(h_k) + B_{aug}(h_k)K_k)z[k] \tag{9}$$

D. Stability Analysis

We have now designed controllers for a system whose sampling period switches between elements of H. This switching behaviour, due to workload variations, can lead to system instability. Therefore, we must *guarantee* stability of the overall system while improving QoC.

Theorem 1. (Stability criterion [16][18]) *Consider A_k to be discrete-time LTI systems. $V(z) = z^T P z$ is the Common Quadratic Lyapunov Function (CQLF) of the systems A_k if there exist $P = P^T > 0$, $Q = Q^T > 0$ and P is the simultaneous solution of the discrete-time Lyapunov equations,*

$$A_k^T P A_k - P = -Q < 0 \tag{10}$$

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

V. Formal Modelling

The first step in SPADe is to derive application and platform models, as shown in Fig. 2. The application model should: i) capture the dynamic behaviour of workload variations, ii) support timing analysis, and iii) support optimal mapping of tasks to the platform. We choose Scenario-Aware DataFlow (SADF) graphs for modelling our application since it satisfies our requirements for the MoC.

A. Capturing Workload Variations

Workload variations during run-time can be statistically analysed and frequently occurring workload scenarios can be classified. However, the exact sequence of occurrence of workload scenarios cannot be determined and can only be assumed to be arbitrary. Hence, it is important for our models to capture workload variations so that we can analyse run-time behaviour and efficiently design system configurations.

To capture workload variations, we need to first identify the parameters in input data (the image frame) that relates to workload. These parameters could be features that are extracted from the image frame like motion vectors. We assume that the information on these parameters is known early on during the frame processing, e.g. from the frame header. For our example, we consider image pre-processing algorithms that define Regions-of-Interest (RoI) on the image plane [21]. Further, we quantify workload, w, based on the number of RoI, i.e. #RoI $= w$.

An image processing algorithm (represented as T_s in Fig. 3) that takes as input pre-processed image frames with RoI information, usually has three main tasks: i) a RoI detection (RoID) task to detect the #RoI; ii) a parallelisable RoI processing (RoIP) task, e.g. a sobel filter to detect the edges in each region; and iii) a RoI merging (RoIM) task to merge the

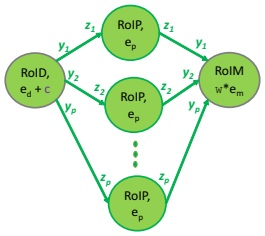

Fig. 5: Scenario-Aware Data Flow (SADF) model of T_s for p processor tiles

processed regions and compute the parameter of interest, e.g. y_L (as defined in Section IV-C). RoID detects the workload w and initiates reconfiguration, if required. The time cost for reconfiguration, c, is then abstracted along with the WCET, e_d, for RoID. e_p and e_m are the WCETs for processing one RoI. The WCET of RoIM is dependent on the image workload. We define the minimum and maximum workload as (w_{min}, w_{max}). For our example, $w_{min} = 1$ and $w_{max} = 6$.

The maximum possible number of concurrent parallel executions of RoIP is determined by the number of processing tiles in the platform. For a platform with p processor tiles, we can model T_s as shown in Fig. 5. Here, #RoIP actors = p, y_i represents workload distribution among the processor tiles and z_i represents whether the code is executed on a processor tile i. For $i = (1, \ldots, p)$, $w \in [w_{min}, w_{max}]$,

$$\begin{aligned} y_i &= \{0, 1, \ldots, \left\lceil \frac{w}{p} \right\rceil\}, \; s.t. \sum_{i=1}^{p} y_i = w \\ z_i &= \begin{cases} 0, \text{if } y_i = 0; \\ 1, \text{otherwise.} \end{cases} \end{aligned} \tag{11}$$

As an example, for a platform with 2 processor tiles, T_s is shown in Fig. 4 and the corresponding parameters are

$$\begin{aligned} &y_1 = \{1, \ldots, \left\lceil \frac{w}{2} \right\rceil\} && z_1 = 1 \\ &y_2 = \{0, 1, \ldots, \left\lfloor \frac{w}{2} \right\rfloor\} \quad ; && z_2 = \begin{cases} 0, & \text{if } y_2 = 0; \\ 1, & \text{otherwise.} \end{cases} \\ &w = y_1 + y_2, \end{aligned}$$

B. Application Model

Our application model is an SADF graph that captures workload variations and models the sequential execution of T_s, T_c and T_a tasks. The WCETs of tasks T_c and T_a (modelled as actors in SADF, see Fig. 4) are e_c and e_a, respectively. Our scenarios are defined based on the workload. A *workload scenario*, S, is defined by #RoI, w, using Eq.(11). We model the time-triggered execution of T_a by introducing a delay, $d(S_i)$. $d(S_i)$ is abstracted along with actor T_c, as shown in Fig. 4, and is computed during the analysis and design phase of SPADe.

C. Platform Model

A platform is modelled as a platform graph as explained in [19]. The image data input from the camera module is available to the platform at a rate of 30 fps. Binding of application to platform refers to the mapping of execution of tasks to the processors, data to memory and communication

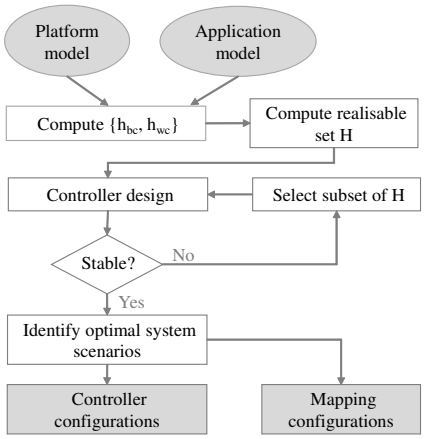

Fig. 6: Analysis and Design Flow

to network interface [19]. A schedule refers to the sequence of execution of tasks mapped on the same processor.

VI. Analysis, Design and Implementation

The goal of this step in SPADe is to identify optimal *system scenarios* and design the system configurations. We cluster workload scenarios into system scenarios that take into account the control aspect. The design flow to realise this is illustrated in Fig. 6.

A. Compute Sampling Period

The first step in analysis is to compute the best-case and worst-case sampling period (h_{bc}, h_{wc}) corresponding to minimum and maximum workload (w_{min}, w_{max}). Our sampling period (h_i) computation is based on the following two crucial observations.

Observation 1: A workload scenario S_i has a (worst-case) sampling period h_i for a chosen platform mapping. A workload scenario sequence $S_i \rightarrow S_j \rightarrow S_j$ implies a sampling period sequence $h_i \rightarrow h_j \rightarrow h_j$.

This observation follows from the fact that the tasks T_s, T_c and T_a are executed sequentially. This helps us in compositional analysis of workload scenarios since we see that the scenario sequence does not affect sampling periods of individual workload scenarios.

Observation 2: For a workload scenario sequence $(S_i)^*$, i.e. $S_i \rightarrow S_i \rightarrow S_i \ldots$, the sampling period h_i of S_i is

$$h_i = \frac{1}{throughput(S_i)} \tag{12}$$

These observations help us to relate the dataflow techniques such as throughput analysis with the control design parameter, sampling period. The computation of the sampling period hence reduces to a throughput analysis problem.

However, 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 want to improve QoC which means shorter settling time (see Eq. 2) and hence maximum throughput (Eq. 12). A shorter settling time can be achieved by having smaller sampling periods [1, 26]. Our objective is then to find maximum throughput for a workload scenario, given the platform allocation and workload. Our 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 platform while maximising throughput can be used. We use the SDF3 design flow [19, 25] as it optimises the resource usage, memory load and communication load for mapping, and embeds state-of-the-art throughput analysis techniques. Algorithm 1 briefly explains the method used in SPADe. Sampling period is then computed using Eq. 12. For our example, we obtain $h_{bc} = 1/\text{maxThr}(w_{min}) = 0.030s$ and $h_{wc} = 1/\text{maxThr}(w_{max}) = 0.078s$.

Algorithm 1: maxThr(w)

input : w, application model, platform model
output: mapping configuration (mc) and its throughput.

construct SDF graph for workload w, using Eq. 11;
run SDF3 flow [19, 25]:
    input - SDF graph for w and platform model;
    output - optimal mappings (bindings + schedules);
$mc \leftarrow$ the mapping with maximum throughput;
throughput $\leftarrow$ throughput of mc;

B. Compute Realisable Set H

The next step is to compute the set of realisable sampling periods, H. The number of image frames that arrive within the time interval (h_{bc}, h_{wc}) determines H. Algorithm 2 outlines the computation of H. For our example, $fps = 30$; $n = 3$, $i = 1$ and $H = \{0.033, 0.066, 0.100\}$.

Algorithm 2: compute H

input : h_{bc}, h_{wc}, fps - camera input rate

$n \leftarrow \lceil h_{wc} * fps \rceil$;
find min(i), s.t. $i \in \mathbb{N} \wedge \frac{i}{fps} \geq h_{bc}$;
$H \leftarrow \{\frac{i}{fps}, \frac{i+1}{fps}, \ldots, \frac{n}{fps}\}$;

C. Controller Design and Stability

After computing the realisable set H, we design stable controllers for each sampling period in H as explained in Section IV-C. Our system can switch between different sampling periods in H due to workload variations and this can lead to system instability. To *guarantee* stability of the overall system, we derive the LMIs from the stabilisation condition (Eq. 10), to analyse for the existence of a CQLF. The analysis equation, Eq. 13, is obtained by performing some operations: i) substitute A_k in Eq. 10 with $A_k = A_{aug}(h_k) + B_{aug}(h_k) * K_k$, ii) apply Schur complement, and iii) left- and right- multiplication by diag(P^{-1}, I) and set $Q = P^{-1}$.

$$\begin{bmatrix} -Q & QA_*^T + QK_k^T B_*^T \\ A_*Q + B_*K_kQ & -Q \end{bmatrix} < 0,\ Q > 0 \tag{13}$$

where $A_* = A_{aug}(h_k)$, $B_* = B_{aug}(h_k)$ for $h_k \in H$. If a solution exists, then the switching subsystems are stable. If a solution does not exist, then we need to find another (subset) H_{ss} that is stable. Algorithm 3 explains our method to choose subset H_{ss}, iteratively for the controller design flow in Fig. 7.

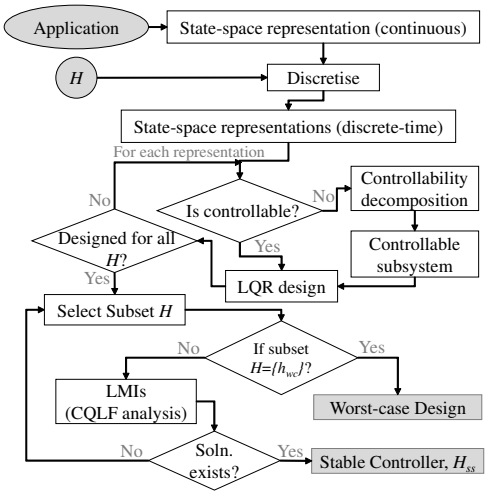

Fig. 7: Controller design

When a CQLF solution does not exist, SPADe reduces to worst-case based design. Thus, the worst-case solution for this algorithm is the worst-case sampling period. For our running example, $H_{ss} = \{0.033, 0.066, 0.100\}$.

Algorithm 3: selectSubsetH(H)

input : H (sorted in ascending order)
output: H_{ss}: subset of H with shortest average h

if *computing for first time* **then**
  $n \leftarrow \mid H \mid$, so that $H = \{h_1, \ldots, h_n\}$;
  find all subsets of H having element h_n, H_s : $H_s \subseteq H \wedge \{h_n \in H_{s_i},\ i \in (1,\ldots,|H_s|) \wedge H_{s_i} \in H_s\}$;
  $f(H_{s_i}), h_{avg_i} \leftarrow \frac{\sum_{j=1}^{|H_{s_i}|} h_j}{|H_{s_i}|},\ h_j \in H_{s_i} \wedge H_{s_i} \in H_s$;
  sort H_s based on h_{avg_i} in ascending order: if $h_{avg_{i,j}}$ are equal, sort based on $\mid H_{s_{i,j}} \mid$;
$H_{ss} \leftarrow H_{s_1}$, first element of H_s
remove H_{s_1} from H_s, $H_s \leftarrow \{H_s - H_{s_1}\}$

A **controller configuration** is a combination of sampling period h, sensor-to-actuator delay τ and feedback gain K.

D. System Scenario Identification

Once we obtain a stable switching subsystem, we have to identify optimal system scenarios. For each sampling period, h_k ($h_k \in H_{ss}$), we should find the optimal mapping and controller configurations. The controller configurations for controllers in H_{ss} are computed in Sec. VI-C. Sampling periods of every workload scenarios are not necessarily realisable (see Sec. VI-B). System scenarios abstract and cluster the workload scenarios with realisable and stable sampling periods, H_{ss}.

Identifying system scenarios involves searching for the maximal workload that fits each h_k. This is derived from Eq. 11 and Eq. 12 that defines a workload scenario, S_i, using workload, w, and computes h_k for S_i. Delay $d(S_i)$ is computed for each workload scenario, S_i, such that the closest realisable h_k is achieved. We also observe that $\mid H_{ss} \mid$ would preferably be small as it is difficult to prove the existence of CQLF

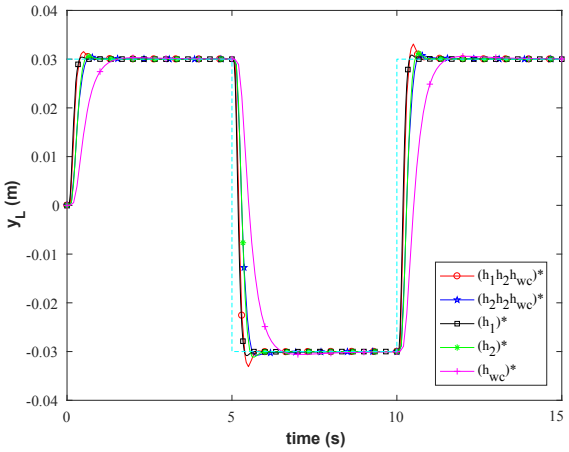

Fig. 8: Controller performance: comparison of switching subsystems with h_{wc}

for a larger number of switching subsystems. Hence, in this work, we do an exhaustive search to identify the maximally fitting workloads, w_k, for each h_k using Algorithm 1. Here, we compute h_i for all workloads $w_i, w_i \in (w_{min}, w_{max})$. Then we update W with the workloads that maximally fit H_{ss}.

System scenarios, SS, are then the workload scenarios defined by maximally fitting workloads, W. Mapping configurations of SS are then the mapping configurations for workload scenarios defined using W and can be obtained using Algorithm 1. A system configuration defines the mapping and controller configuration of a system scenario. For our example, $W = \{2, 4, 6\}$ for $H_{ss} = \{0.033, 0.066, 0.100\}$, i.e. 2 RoI fit the 0.033 sampling period and so on.

E. Reconfiguration Mechanism

During run-time, for every input image frame, we compute the workload (image pre-processing) and choose the correct system scenario, i.e. the system scenario with the closest (maximal) workload. System configurations of system scenarios are stored in a look-up table (LUT). A scheduler then reconfigures the mapping, time-triggering of T_a and controller 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, c, in the actor RoID of Fig. 5 (see Section V-A).

VII. Results

The model we use for vision-based lateral control of a vehicle is derived from [20]. The state-space representation of the vehicle and the control objective are explained in Sec. VI-C. We simulate the controller performance for different switching sequences of $H_{ss} = \{h_1,\ h_2,\ , h_{wc}\} = \{0.033,\ 0.066, 0.100\}$ as shown in Fig. 8. We observe that our switching designs of SPADe (plot $(h_1 h_2 h_{wc})^*$ and $(h_2 h_2 h_{wc})^*$) settle faster (and hence have better QoC) than the worst-case sampling period based design (see plot $(h_{wc})^*$) in Fig. 8. An example switching sequence is illustrated in Fig. 9 for $W = \{2,\ 4,\ 6\}$. We see that the effective resource

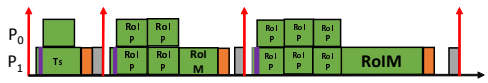

Fig. 9: Switching sequence $(h_1 h_2 h_{wc})^*$

utilisation for each sampling period is maximised with respect to the worst-case based design in Fig. 1.

VIII. Conclusion

We presented a structured (co-)design flow that considers application dependencies, platform settings and control parameters for an efficient design and implementation of an IBC system. Our Scenario- and Platform-Aware Design (SPADe) approach optimises QoC and maximises the effective resource utilisation for a given platform allocation. Our SPADe approach performs proficiently when the image processing algorithm is known, i.e. it is a white/gray box approach. In this case, SPADe assumes that information on parameters that relate to workload is known or can be computed during the frame pre-processing. We can then efficiently design mapping and controller configurations and a reconfiguration mechanism.

Future work is to study how we can extend our approach when the image processing algorithm is a black box by considering reconfigurable pipelined control, where multiple image processing tasks are pipelined to hide the image processing latency. Analysis is also required to evaluate whether pipelining or parallelization like in SPADe or a combination of both would be better for control performance. Further, our current approach addresses a single application setting. It is interesting to extend the SPADe approach for multiple applications sharing a platform allocation.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875.

References

[1] E.P. van Horssen, "Data-intensive feedback control: switched systems analysis and design" PhD Dissertation, Eindhoven University of Technology, 2018.

[2] P. Corke, "Robotics, Vision and Control: Fundamental Algorithms in MATLAB," Second, Completely Revised. Vol. 118. Springer, 2017.

[3] E. Malis, "Survey of vision-based robot control," in European Naval Ship Design, ENSIETA, 2002.

[4] N. Oda, M. Ito, and M. Shibata, "Vision-based motion control for robotic systems," IEEJ Transactions on Electrical and Electronic Engineering, vol. 4, no. 2, 2009.

[5] J. Elfring, et al., "Effective world modeling: Multisensor data fusion methodology for automated driving," Sensors, vol. 16, no. 10, 2016.

[6] S. D. Pendleton, et al., "Perception, planning, control, and coordination for autonomous vehicles," Machines, vol. 5, no. 1, 2017.

[7] K. Bengler, et al., "Three decades of driver assistance systems: Review and future perspectives." IEEE Intelligent Transportation Systems Magazine 6.4, 2014.

[8] FEI, "An Introduction to electron microscopy," ISBN 978-0-578-06276-1, 2010.

[9] I. Chakraborty, et al., "Compensating for time-varying input and state delays inherent to image-based control systems," in ACC, 2016.

[10] S. Adyanthaya, et al., "Robustness analysis of multiprocessor schedules," in SAMOS XIV, pages 9-17, 2014.

[11] S.V. Gheorghita, et al., "System-Scenario-based Design of Dynamic Embedded Systems," in ToDAES, 2009.

[12] Y. Wang, et al., "Multirate estimation and control of body slip angle for electric vehicles based on onboard vision system," IEEE Trans. on Industrial Electronics, 2014.

[13] A. Kawamura, et al., "Robust visual servoing for object manipulation with large time-delays of visual information," in Proc. IROS, 2012.

[14] G. Macesanu and V. Comnac, "A time-delay control approach for a stereo vision based human-machine interaction system," J. Intell Robot Syst., 2014.

[15] R. Medina, et al., "Reconfigurable pipelined sensing for image-based control," In SIES, 2016.

[16] Z. Sun, S. S. Ge, "Stability theory of switched dynamical systems." Springer Science & Business Media, 2011.

[17] A. Y. Bhave, B. H. Krogh, "Performance bounds on state-feedback controllers with network delay," in CDC, 2008.

[18] O. Mason, R. Shorten, "On common quadratic Lyapunov functions for stable discrete-time LTI systems," IMA Journal of Applied Mathematics 69.3, 2004.

[19] S. Stuijk, "Predictable mapping of streaming applications on multiprocessors," Ph.D. dissertation, Eindhoven University of Technology, 2007, dOI:10.6100/IR630223.

[20] C. J. Taylor, et al., "A comparative study of vision-based lateral control strategies for autonomous highway driving," The International Journal of Robotics Research 18.5: 442-453, 1999.

[21] Hillel A.B., et al., "Recent progress in road and lane detection: a survey," Machine vision and applications. 25(3): 727-45, 2014.

[22] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," Proceedings of the IEEE, 75.9, 1987.

[23] S. S. Bhattacharyya, P. K. Murthy and E. A. Lee, "Software synthesis from dataflow graphs," Springer Science & Business Media (Vol. 360), 2012.

[24] B. D. Theelen, et al., "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in MEMOCODE, 2006.

[25] S. Stuijk, M.C.W. Geilen and T. Basten, "SDF3: SDF for free," In ACSD 2006.

[26] J. Valencia, D. Goswami, K. Goossens, "Composable platform-aware embedded control systems on a multicore architecture," In DSD, pp. 502 - 509, 2015.

[27] R.C. Dorf and R.H. Bishop, "Modern control systems," Pearson, 2011.

[28] M.M. Rosenbrock, "State-Space and Multivariable Theory," John Wiley, 1970.