Reconfigurable Pipelined Control Systems

Róbinson Medina Sanchez
TNO, Powertrains, The Netherlands

Shayan Tabatabaei Nikkhah, **Dip Goswami**, **Maurice Heemels**, **and Sander Stuijk**
Eindhoven University of Technology, Eindhoven, The Netherlands

Twan Basten
Eindhoven University of Technology, Eindhoven, The Netherlands
and ESI, TNO Eindhven, The Netherlands

Editor's notes:
As sensor data processing, e.g., from video cameras, is becoming more compute intensive, it is also introducing higher sensor-to-actuator delays. This article studies cross-layer schemes that co-design a controller together with sensor data processing in a pipelined fashion on multicore embedded processors.
—Samarjit Chakraborty, University of North Carolina at Chapel Hill

Intensive sensor-data processing in a control loop occurs in many application domains including advanced driver assistance systems (ADAs) [1] and vision-guided control [2]. Such sensing technologies allow one to extract a rich set of information from the environment and is instrumental to develop reliable functionality in various modern systems. An example is image processing in the loop where a camera and an image-processing algorithm are used for sensing [3]. The key challenge in designing such systems is to deal with the long sensing delay resulting from the heavy computation demand for sensor-data processing.

Strategies for coping with a long sensing delay are reported both in the embedded systems [4], [5] and in the control theory literature [2], [6]. In [6], sensing delay is compensated by advanced estimation techniques and in [2] multirate actuation strategies are used. These strategies heavily rely on the system model and are vulnerable to modeling errors. Embedded system approaches aim to reduce the data-processing latency by creating parallel implementations of the algorithms in specialized hardware such as GPUs [4] or FPGAs [5]. These strategies require a great effort to speed up the data-processing algorithms and the final latency may still not be acceptable for safety-critical applications.

A control loop typically requires to perform sensing, control computation, and actuation operations periodically to regulate the behavior of a dynamic system. In a classical implementation (Figure 1), these operations are performed sequentially in which long sensor-data processing would lead to a long execution time. A sampling period h is chosen at least equal to the total execution time τ of the loop. A camera, for example, can capture image frames with a fixed period f_h. Typically, the camera frame rate can be much higher than the rate at which the frames are processed, i.e., $f_h \ll h$. For instance, in Figure 1, $h = 6f_h$; only every sixth frame is processed due to the long data-processing time. Such a long sampling period can lead to a degraded control performance

Digital Object Identifier 10.1109/MDAT.2020.3006803
Date of publication: 3 July 2020; date of current version: 28 September 2021.

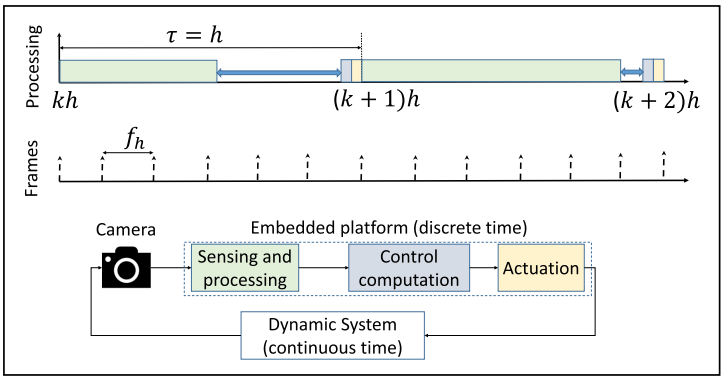

Figure 1. Data-intensive feedback control systems. The execution time of sensor-data processing may vary. Control computation and actuation are time triggered, taking into account the worst case sensor-data processing time, to ensure a constant sampling period. Color code: green for sensing, blue for computing and yellow for actuating.

or even instability and is often unacceptable in safety-critical domains.

A potential solution may be found in cross-layer design taking into account both controller design and its embedded implementation. The key idea is to increase the rate of sensor-data processing by processing frames in parallel on multiple cores. In this way, the sensor processing rate can be increased while the sensing delay remains unaffected. To realize such a pipelined implementation, we need scheduling and reconfiguration mechanisms to match model-level assumptions made in the control design. Furthermore, we need control algorithms taking into account implementation-level artifacts to deal with the long delay and to translate a higher sensing rate to a better performance. Such cross-layer design problems require novel techniques to handle challenges across multiple layers that are usually dealt with in isolation.

Why pipelined control (PC)? PC systems (illustrated in Figure 3a) can outperform a sequential implementation as was shown in [7]. In a pipelined loop, the time between two consecutive completions of sensor-data processing is reduced thereby essentially shortening the sampling period, while the delay is unaltered and thus, remains long. This is illustrated in Figure 3, which shows an example with three control cores. The sampling period is reduced to $2f_h$ compared to the sequential case with sampling period $6f_h$. The method is particularly suitable when consecutive data samples (images, or any other type of data) can be processed independently. In general, there is a tradeoff between the possible overhead caused by independence between data samples and the (performance) gains of pipelining. PC is enabled by current trends in computer architecture integrating multi/many-cores, GPUs, and FPGAs in embedded systems.

Why reconfigurable PC? Parallel processing platforms may be shared among multiple applications for cost reasons. Availability of cores may change at runtime depending on the activation of (e.g., sporadic) tasks. Statically allocating cores to the pipelined loops limits interference between applications, however, would lead to a small number of available cores for the control loop. An alternative is to dynamically allocate cores to the sensor-data processing depending on their availability. Figure 5 shows an example where the number of available cores temporarily changes from three to two (and the system has to reconfigure accordingly). If the pipelined system can run most of the time with three cores while it may only occasionally need to run with two cores, a static allocation would lead to using only two control cores. In contrast, reconfigurable pipelined control (RPC) could use most of the time three cores and thereby has the potential to outperform static PC in a long run, if designed and implemented properly. The RPC design and analysis aspects are studied in [8]. The current work focuses on how the proposed method can be implemented on embedded platforms. Our implementation is based on a proof-of-concept implementation architecture for PC in [3].

In an RPC system, we need to take into account the implementation-layer timing artifacts in the control model, design, and optimization. Moreover, we need to schedule the control cores to realize the temporal behavior assumed at model-level and in control design. Therefore, an RPC system requires a reconfiguration mechanism that respects timing assumptions made at the model-level under all configurations as well as a control design technique that guarantees stability and an improved performance under possible configuration switching. In this work, we emphasize the implementation aspects of RPC systems and illustrate these on an experimental setup.

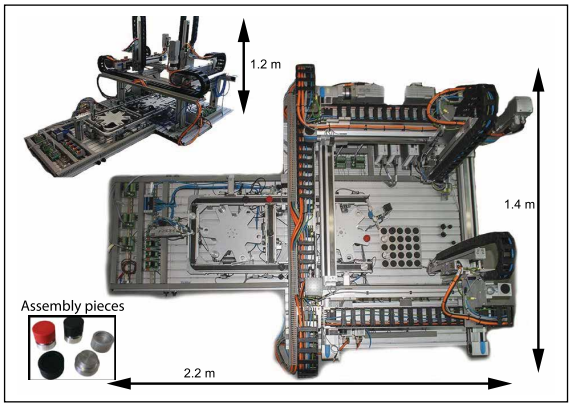

Figure 2. xCPS platform [3].

System setup

This section introduces the control problem at hand.

Motivating application: xCPS

The explore cyber–physical systems (xCPSs) platform (Figure 2) is an industrial assembly line testbed [9]. The machine assembles pieces that come in two shapes: 1) lower and 2) upper parts. Conveyor belts move the assembly pieces through the machine actuators: a turner, multiple stoppers, and a pick-and-place unit, among others. Regulating the speed and positions of the assembly blocks is crucial for guaranteeing the productivity of the assembly process. To do so, a camera running at 60 frames/s and an image-processing algorithm based on the Hough transform for circles are used to measure the speed and position of the assembly blocks. The control operations and sensor processing are performed on a four-core embedded board. This is a representative system capturing a number of common challenges in many real-life applications that use vision-based processing in the feedback loop [1].

Feedback control systems

A wide class of dynamic systems, including the motion system related to the assembly blocks mentioned above, can be described by the linear state–space model

$$\dot{x}(t) = A_c x(t) + B_c u(t) \qquad (1)$$
$$y(t) = C_c x(t)$$

where $A_c \in \mathbb{R}^{n\times n}$, $B_c \in \mathbb{R}^{n\times m}$, and $C_c \in \mathbb{R}^{p\times n}$ are system, input, and output matrices, respectively, and $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the control input used for actuation, and $y(t) \in \mathbb{R}^p$ is the output to be regulated, at time $t \in \mathbb{R}_{\geq 0}$. With sensor-to-actuator delay τ and zero-order-hold mechanism with sampling period $h \in \mathbb{R}_{\geq 0}$, $u(t)$ becomes a piecewise constant signal, being constant in the intervals $[kh+\tau, (k+1)h+\tau)$ for $k \in \mathbb{Z}_{\geq 0}$. A PC system is relevant when τ is long as motivated in the introduction. We assume that the full state is available for feedback although controller-output based design could be considered as well.

Example 1. The model of an xCPS conveyor belt has two states: $x = [x_1\ x_2]^T$ (position and velocity), and one input u (motor voltage), one output $y = x_1$ (position of the belt). The system matrices are given by

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

Sensor-to-actuation loop execution

A control or sensor-to-actuator loop is executed by periodically performing sensing, control computation, and actuation operations. For the system in (1), the sensing, reading, and processing of system state $x(t)$, starts periodically at $t = kh$, $k \in \mathbb{Z}_{\geq 0}$. The sampling period h is the interval between two consecutive activations of the sensing operations that require image frames for processing. A camera captures the images at discrete instants $t_f \in \{if_h | i \in \mathbb{Z}_{\geq 0}\}$, where f_h is the frame at discrete intervals, is chosen to be integer divisible by arrival period. In xCPS

$$f_h = 1/60 \text{ s} = 16.67 \text{ ms}.$$

We align the start of a sensing operation with t_f (i.e., frame arrival times) as shown in Figure 3; hence, h is an integer multiple of f_h. Sensor processing is followed by control computation and actuation operations, which generally take short and nearly constant time for execution. Sensing takes much longer time due to the computationally heavy processing, that is

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

where τ_s, τ_c, and τ_a are the worst case execution times of sensor-data processing, control computation, and actuation operations, respectively. The total (worst-case) execution time τ_t of a sensor-to-actuator loop is thus given by

$$\tau_t = \tau_s + \tau_c + \tau_a. \qquad (2)$$

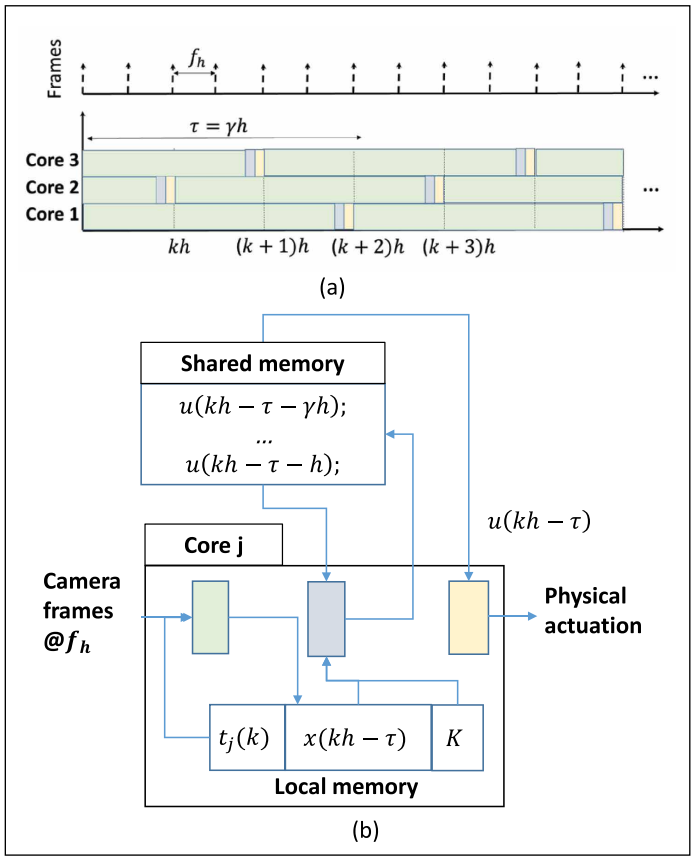

Figure 3. PC systems: (a) timing and (b) implementation architecture.

Due to alignment of the start of a sensor operation with t_f, the effective sensor-to-actuator execution time τ is given by

$$\tau = \left\lceil \frac{\tau_t}{f_h} \right\rceil f_h. \quad (3)$$

Clearly, $\tau \geq \tau_t$. While the sensor processing has variable execution time (see the blue arrows in Figure 1), time-triggered activations result in a constant input-to-output delay τ (an integer multiple of f_h).

Example 2. A camera and an image-processing algorithm are used as a sensor for measuring the states x_1 and x_2 in Example 1. The total worst case execution time τ_t equals 84 ms with f_h = 16.67 ms, $\tau = 6f_h$ = 100 ms following (3). That is, every sixth frame is processed by a core.

PC systems

PC is realized by executing sensor-to-actuator loops in parallel in γ globally synchronous cores (see Figure 3a for $\gamma = 3$) and a camera with a fixed frame rate. The choice of γ is made based on the requirement on sampling period h [which depends on the system dynamics (1)] and estimation of τ on a given platform. A proof-of-concept implementation architecture in [3] uses time-triggered activations. The cores are denoted by Core 1 to Core γ, where Core 1 is referred to as the base core. We denote the start of the sensor-to-actuator loop for the kth sample in Core j as $t_j(k)$. The pipelining is initiated by the base core at $t_1(0) = 0$. The triggering offset between Core $j+1$ and Core j is related as

$$t_{j+1}(k+1) = t_j(k) + h \quad (4)$$

where h is the sampling period given by

$$h = \frac{\tau}{\gamma}. \quad (5)$$

Furthermore, we trigger the loop on Core j as

$$t_j(k+\gamma) = t_j(k) + \tau. \quad (6)$$

In essence, the duration τ is divided into γ samples of length h. Clearly, a larger γ gives a shorter sampling period h. To align the control operations with the availability of the frames at discrete intervals, $\left\lceil \frac{\tau_t}{f_h} \right\rceil$ is chosen to be integer divisible by γ. Hence, a set of finite configurations are considered.

Example 3. We know from Example 2 that $\tau = 6f_h$, and we have a quad-core platform. With $\gamma = 2$, the sampling period $h = 3f_h$. Similarly, $\gamma = 3$ gives $h = 2f_h$. The choice $\gamma = 4$ causes miss alignment with the frame arrival and, therefore, is not used here. Hence, $\gamma \in \{1, 2, 3\}$ is admissible in the example.

Discretized pipelined model

We discretize the continuous-time system (1) with sampling period h and sensor-to-actuator delay τ:

$$x((k+1)h) = A_d x(kh) + B_d u(kh - \gamma h) \quad (7)$$

where A_d and B_d are the discretized state and input matrices [8]. The control input $u(t)$ at $t = kh$ uses γh time units (or γ frames) old sensing information in any sampling interval kh to $(k+1)h$ due to the sensor-to-actuator delay τ. This is reflected in (7) as the delayed input $u(kh - \gamma h)$.

For designing the delayed control input $u(kh - \gamma h)$, one design option is to transform system (7) into standard nondelayed form and apply standard control design techniques. For this purpose, we define a new system state vector including γ old input signals as

$$z(kh) = [x^T(kh)\ u(kh - \gamma h) \cdots u(kh - h)]^T. \quad (8)$$

Using the state definition in (8), the system model (7) is transformed into the following nondelayed form:

$$z(kh + h) = \Phi_d z(kh) + \Gamma_d u(kh) \quad (9)$$
$$y(kh) = C_d z(kh)$$

where Φ_d, Γ_d, and C_d are the corresponding augmented discretized matrices and z is the new augmented state vector. See [8] for details.

Example 4. From Example 2, we know that $\tau = 100$ ms and $n = 2$ (two states). Consider $\gamma = 2$ and the resulting sampling period $h = 3f_h$. The augmented state vector of (8) has $(n + \gamma)$ states and is given by $z(kh) = [x_1(kh)\ x_2(kh)\ u(kh - 2_h)\ u(kh - h)]^T$ with measured states $x(kh)$ at $t = kh$ and two older input signals.

PC law and performance

System (9) is in standard discrete-time form for which standard discrete-time control design techniques such as LQR [3] can be used in the form

$$u(kh) = Kz(kh) \quad (10)$$

where K is the state feedback gain to be designed. We apply $u(t) = 0$ for $t < \tau$ since we need to wait for the first sample to be processed before the first feedback-based actuation can take place. As a performance metric, we consider the settling time, which is defined here as the time that the system output takes to reach and stay within a bound of 2% around the reference or zero. Other metrics can also be considered depending on the system requirements.

Example 5. Let us continue with $\tau = 100$ ms $= 6f_h$, $\gamma = 2$ and the resulting sampling period $h = 3f_h$. Control law (10) uses the system state vector $z(kh)$ defined in (8), which has both measured state $x(kh)$ and γ old (stored) input signals. From (7), old state vector $z(kh - \tau)$ is used by (10) in the interval $[kh, (k+1)h)$, i.e., the "sensing delay" is modeled as the "actuator delay"

$$u(kh - \gamma h) = K \begin{bmatrix} x_1(kh - 6f_h) \\ x_2(kh - 6f_h) \\ u(kh - 12f_h) \\ u(kh - 9f_h) \end{bmatrix}.$$

Clearly, the measured signal x is six frames old and there are other older input signals in the control law. K is designed using the technique presented in [3] leading to

$$K = [-30.9\ -7.3\ -2.7\ -1.9]$$

which results in a settling time of 123.5 ms.

The settling time improves with a higher number of PC cores [3]. A settling time below 120 ms is achieved with $\gamma = 3$, whereas a sequential implementation with $\gamma = 1$ gives a settling time larger than 220 ms.

Implementation architecture for PC

Our implementation architecture for the PC is shown in Figure 3b, based on [3]. A sensor-to-actuator loop is triggered as per (4) and (6) and it has an (effective) constant execution time of τ, which implies that the processed states are $x(kh - \tau)$ or τ time old. The sensor-data processing is performed on a single image to obtain $x(kh - \tau)$ and stored in local memory (no data dependency between the sensing pipes). The computing task uses $x(kh - \tau)$ and the feedback gain K from the local memory since they are not dependent on other cores. It uses γ old input signals from shared (among the γ control cores) memory. Access to the shared memory is realized with a predictable memory controller (by nonoverlapping allocation) reported in [10]. Overall, the control tasks have nearly constant execution times under this predictable time-triggered architecture.

Reconfigurable PC systems

Figure 4 illustrates the idea of RPC systems. The system may run either in the maximal configuration (MC) or in a reduced configuration (RC) depending on the availability of the cores. A MC uses γ_{MC} control pipes and the feedback gain K_{MC} in the control law (10). The sampling period h_{MC} in the MC is given by

$$h_{\mathrm{MC}} = \frac{\tau}{\gamma_{\mathrm{MC}}}. \quad (11)$$

Stability and performance of the MC depend on the closed-loop system matrix $(\Phi_{\mathrm{MC}} + \Gamma_{\mathrm{MC}}\ K_{\mathrm{MC}})$. Similarly, an RC uses $\gamma_{\mathrm{RC},i}$ control pipes (which is

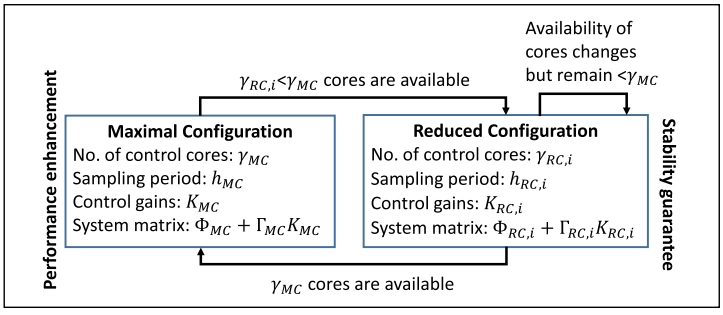

Figure 4. Reconfiguration PC systems.

smaller than γ_{MC}) and the feedback gain $K_{RC,i}$ in the control law (10). The sampling period $h_{RC,i}$ in an RC is given by

$$h_{RC,i} = \frac{\tau}{\gamma_{RC,i}}. \quad (12)$$

The closed-loop matrix ($\Phi_{RC,i} + \Gamma_{RC,i} K_{RC,i}$) dictates the stability and performance in an RC. There may be multiple RCs.

Example 6. For the system introduced in the "Motivating application: xCPS" section, the possible configurations are given by $\gamma \in \{1, 2, 3\}$. Here, $\gamma_{MC} = 3$ and there are two RCs, e.g., $\gamma_{RC,1} = 1$ and $\gamma_{RC,2} = 2$.

Reconfiguration controller design

We design the feedback gains K_{MC} and $K_{RC,i}$ for overall system stability and performance optimization following [8]. As per the definition of the augmented state in (8), the state vector differs in dimension in various configurations. Stability analysis of the switched systems can be performed among the systems of identical dimension. We match the dimension in the configurations by augmenting exactly γ_{MC} old input signals as shown in (8).

Example 7. Recall Example 2. $\tau = 100$ ms. The MC $\gamma_{MC} = 3$. The resulting sampling period $h_{MC} = 2f_h$. The augmented state vector has $(n + \gamma_{MC}) = 5$ states

$$z(kh_{MC}) = \begin{bmatrix} x_1(kh_{MC}) \\ x_2(kh_{MC}) \\ u((k-3)h_{MC}) \\ u((k-2)h_{MC}) \\ u((k-1)h_{MC}) \end{bmatrix} = \begin{bmatrix} x_1(kh_{MC}) \\ x_2(kh_{MC}) \\ u(kh_{MC} - 6f_h) \\ u(kh_{MC} - 4f_h) \\ u(kh_{MC} - 2f_h) \end{bmatrix}.$$

Next, we consider the RC with $\gamma_{RC,i} = 2$. The resulting sampling period $h_{RC,i} = 3f_h$. The augmented state vector has $(n + \gamma_{MC}) = 5$ states and is given by

$$z(kh_{RC,i}) = \begin{bmatrix} x_1(kh_{RC,i}) \\ x_2(kh_{RC,i}) \\ u((k-3)h_{RC,i}) \\ u((k-2)h_{RC,i}) \\ u((k-1)h_{RC,i}) \end{bmatrix} = \begin{bmatrix} x_1(kh_{RC,i}) \\ x_2(kh_{RC,i}) \\ u(kh_{RC,i} - 9f_h) \\ u(kh_{RC,i} - 6f_h) \\ u(kh_{RC,i} - 3f_h) \end{bmatrix}$$

It can be seen that an RC uses older input signals compared to the MC.

Reconfiguration mechanism

We consider $\gamma \in \{\gamma_{MC}, \gamma_{RC,i}\}$ globally synchronous cores and a camera with a fixed frame rate. Core 1 is the base core, which initiates the pipelined sensing at $t_1(0) = 0$ and continues at least till all the γ cores complete their first sensor-to-actuation loop as per (4) and (6) before any configuration switching occurs. The reconfiguration process is initiated by the base core at $t_{init} = t_1(k)$ (at a triggering instant of a sensor-to-actuator loop at the base core), while actual switching of the feedback gains (i.e., K_{MC} and $K_{RC,i}$) and sampling periods (i.e., h_{MC} and $h_{RC,i}$) occurs at cores

$$t_{sw} = t_{init} + \tau.$$

The switching between sampling periods needs offset adjustments ($t_j(k)$ in Core j) for which the reconfiguration process must be initiated at least τ time earlier at t_{init}. The offsets are computed as

$$t_j(k + j - 1) = t_{init} + \Delta j. \quad (13)$$

Δj (offset adjustment in Core j) differs in different configuration switching.

Maximal to RC switching: Core 1 to Core $\gamma_{RC,i}$ continue in the RC while the other ($\gamma_{MC} - \gamma_{RC,i}$) are freed

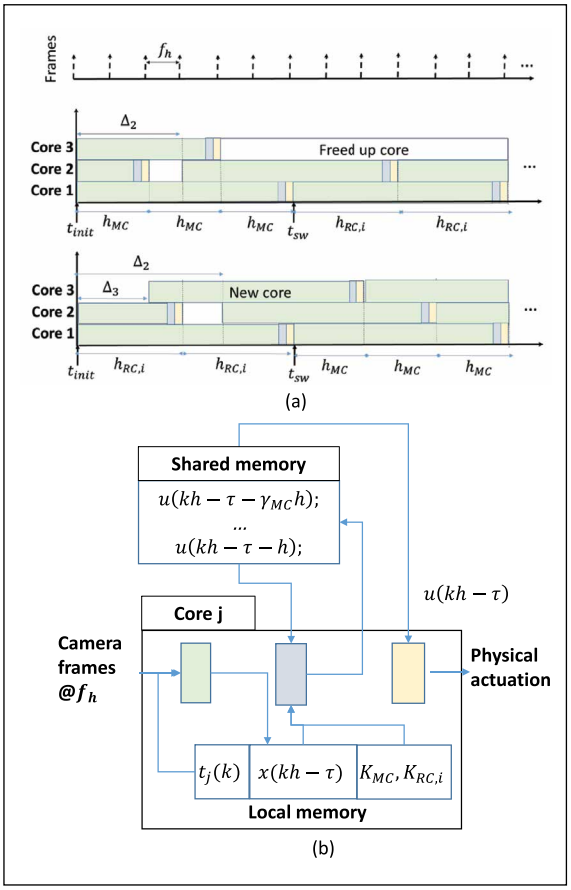

Figure 5. Reconfigurable PC: (a) timing and (b) implementation architecture.

up after completion of an ongoing sensor-to-actuation loop. With $t_{\text{init}} = t_1(k)$, the offset adjustments are computed as

$$\Delta j = (j-1)h_{\text{RC},i}, j \in \{1..\gamma_{\text{RC},i}\}. \quad (14)$$

Figure 5a shows an example reconfiguration from $\gamma_{\text{MC}} = 3$ to $\gamma_{\text{RC},i} = 2$; $\Delta_1 = 0$ and $\Delta_2 = 3f_h$.

Reduced to MC switching: $(\gamma_{\text{MC}} - \gamma_{\text{RC},i})$ new cores are activated. With $t_{\text{init}} = t_1(k)$, the offset adjustments are computed as

$$\begin{aligned}
&\Delta_1 = 0,\\
&\Delta_j = (\gamma_{\text{MC}} - \gamma_{\text{RC},i})h_{\text{MC}} + (j-1)h_{\text{MC}},\ j \in \{2..\gamma_{\text{RC},i}\}\\
&\Delta_j = (j - \gamma_{\text{RC},i})h_{\text{MC}},\ j \in \{\gamma_{\text{RC},i}+1..\gamma_{\text{MC}}\}. \quad (15)
\end{aligned}$$

Figure 5a shows an example reconfiguration from $\gamma_{\text{RC},i} = 2$ to $\gamma_{\text{MC}} = 3$; $\Delta_1 = 0$, $\Delta_2 = 4f_h$, and $\Delta_3 = 2f_h$.

Implementation architecture for RPC

The basic architecture for RPC is the one used for PC (Figure 3b). The stored data needs RPC-specific updates: 1) the feedback gains and $t_j(k)$ updated with the offset adjustments Δj should be stored in the local memory and 2) the shared memory needs to store γ_{MC} input signals. The other important ingredient is the reconfiguration mechanism, which realizes switching between the configurations at runtime using techniques proposed in [11]. In this method, control codes for the sensor-to-actuation loop are already loaded on γ_{MC} cores. Activating/freeing up a core boils down to activating/deactivating a preloaded code instance since the sensor-data processing is performed independently in each core. This also implies that this mechanism has no context switch overhead and has a negligible reconfiguration overhead [11].

A PC IMPLEMENTATION can outperform traditional sequential implementation by executing sensor data-processing in parallel in multiple cores. For example, in the xCPS system, a PC with two (and three) cores achieves a settling time of 123.5 ms (and around 115 ms) as opposed to a sequential implementation which achieves a settling time 220 ms. An RPC switches between available configurations based on available cores can achieve a performance close to the one in the MC in the long run. In the xCPS system, for example, the RPC design achieves a settling time close to 115 ms. We presented how such an RPC scheme can be implemented on an embedded architecture. ■

Acknowledgments

This work was supported in part by the H2020 projects FitOptiVis under Grant 783162 and in part by I-MECH under Grant 737453.

References

[1] K. Jo et al., "Development of autonomous car—Part I: Distributed system architecture and development process," *IEEE Trans. Ind. Electron.*, vol. 61, no. 12, pp. 7131–7140, Dec. 2014.

[2] A. Kawamura et al., "Robust visual servoing for object manipulation with large time-delays of visual information," in *Proc. IROS*, Oct. 2012, pp. 4797–4803.

[3] R. Medina et al., "Designing a controller with image-based pipelined sensing and additive uncertainties," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 3, pp. 1–26, 2019.

[4] R. Agrawal et al., "A GPU based real-time CUDA implementation for obtaining visual saliency," in *Proc. ICVGIP*, 2014, pp. 1–8.

[5] S. Kestur et al., "Emulating mammalian vision on reconfigurable hardware," in *Proc. FCCM*, 2012, pp. 141–148.

[6] H. Fujimoto, "Visual servoing of 6 DOF manipulator by multirate control with depth identification," in *Proc. CDC*, 2003, pp. 5408–5413.

[7] P. Krautgartner and M. Vincze, "Performance evaluation of vision-based control tasks," in *Proc. ICRA*, May 1998, pp. 2315–2320.

[8] R. Medina et al., "Reconfigurable pipelined sensing for image-based control," in *Proc. SIES*, 2016, pp. 1–8.

[9] S. Adyanthaya et al., "xCPS: A tool to explore cyber physical systems," *ACM SIGBED Rev.*, vol. 14, no. 1, pp. 81–95, 2016.

[10] S. Goossens et al., "A reconfigurable real-time SDRAM controller for mixed time-criticality systems," in *Proc. CODES+ISSS*, 2013, pp. 1–10.

[11] L. Gantel et al., "Multiprocessor task migration implementation in a reconfigurable platform," in *Proc. Int. Conf. Reconfig. Comput. (FPGAs)*, 2009, pp. 362–367.

Róbinson Medina Sanchez currently works as a Scientist Innovator at TNO-automotive. His research interests include applied control, automotive systems, and embedded systems. He has an MSc in industrial control engineering from Ghent University, Ghent, Belgium and the University of Ibagu, Ibagu, Colombia (2012).

Shayan Tabatabaei Nikkhah is currently pursuing a PhD in electrical engineering with the Eindhoven University of Technology, Eindhoven, The Netherlands. His research interests include real-time systems, quality and resource management, and approximate computing. He has an MSc in electrical engineering from the University of Tehran, Tehran, Iran (2018).

Dip Goswami is an Assistant Professor with the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. His research interest includes design and analysis for embedded control systems. He has a PhD from the National University of Singapore, Singapore (2009). He is a member if IEEE.

Maurice Heemels is a Professor with the Eindhoven University of Technology, Eindhoven, The Netherlands. He is a Fellow of IEEE.

Sander Stuijk is an Associate Professor with the Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. His research interest includes design trajectories for embedded signal processing application. He has a PhD from the Eindhoven University of Technology (2007).

Twan Basten is a Professor of the Electronic Systems Group, Eindhoven University of Technology, Eindhoven, The Netherlands. His research interest includes model-driven design of embedded and cyber–physical systems. He is a Senior Member of IEEE and a Life Member of ACM.

■ Direct questions and comments about this article to Dip Goswami, Electrical Engineering Department, Eindhoven University of Technology, 5600 Eindhoven, The Netherlands; d.goswami@tue.nl.