Reconfigurable pipelined sensing for image-based control

Róbinson Medina*, Sander Stuijk*, Dip Goswami*, Twan Basten*†
*Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands
†TNO Embedded Systems Innovation, Eindhoven, The Netherlands

***Abstract*—Image-based control systems are becoming common in domains such as robotics, healthcare and industrial automation. Coping with a long sample period because of the latency of the image processing algorithm is an open challenge. Modern multi-core platforms allow to address this challenge by pipelining the sensing algorithm. Often, such systems share the resources with other tasks. We show that the performance of an image-based controller can be improved by pipelining the image processing algorithm on unallocated cores. It can be further improved by dynamically allocating (i.e. reconfiguring) cores that are temporarily not used by other tasks to the sensing pipeline. We present a state-based modelling strategy for pipelined and reconfigurable pipelined sensing. We introduce a control design strategy for reconfigurable pipelined systems that assures stability and shows improvement in the control performance.**

Keywords*—Image-based control, pipelined sensing control, reconfigurable pipelined control.

I. Introduction

An Image-Based Control (IBC) system uses a camera and an image processing algorithm to obtain the sensing information. Such controllers are becoming common in applications such as ADAS (Advanced Driver Assistance Systems) [1], [2] and visual servo control [3]. In IBC, the execution time of the image processing algorithm introduces a *sensing delay* in the control loop. In a classical implementation (See Fig. 1a), the sensing, control computation, and actuating tasks are executed sequentially, forcing the system to have a sample period larger than the sensing delay. The control design becomes challenging when the sample period is much larger than its recommended value due to the sensing delay. A long sample period degrades control performance or may even cause system instability [4].

Nowadays, it is common to use resources with parallel processing capabilities (such as multi-core systems, GPUs, and FPGAs) to execute the IBC together with other tasks in the system. The sensing algorithm can then be parallelized, pipelined (e.g. Fig. 1b) and/or dynamically reconfigured on the available resources. Parallelizing the sensing algorithm is not always straightforward and it could complicate the resource allocation. Our work therefore explores the options to pipeline and dynamically reconfigure the sensing algorithm. These techniques can be applied relatively straightforwardly to any sensing algorithm.

In a pipelined implementation, multiple instances of the image processing algorithm run in parallel in a pipelined fashion. The time between two consecutive completions of sensor data is reduced compared to a sequential execution, decreasing the sample period. It is known that pipelined sensing control is capable of improving the controller performance in visual servo-positioning systems [5]–[7]. The increased usage of multi-core platforms makes it interesting to explore this potential improvement. It is moreover common that applications that use Image-Based Control share resources with multiple sporadic tasks. For example, in ADAS other tasks such as the air conditioning, the alarm system or the GPS system may run on the same resources. These resources can then be dynamically assigned (i.e. reconfigured) between the image processing algorithm and sporadic tasks (e.g. Fig. 1c). This reconfiguration potentially leads to a further improvement in the quality of control.

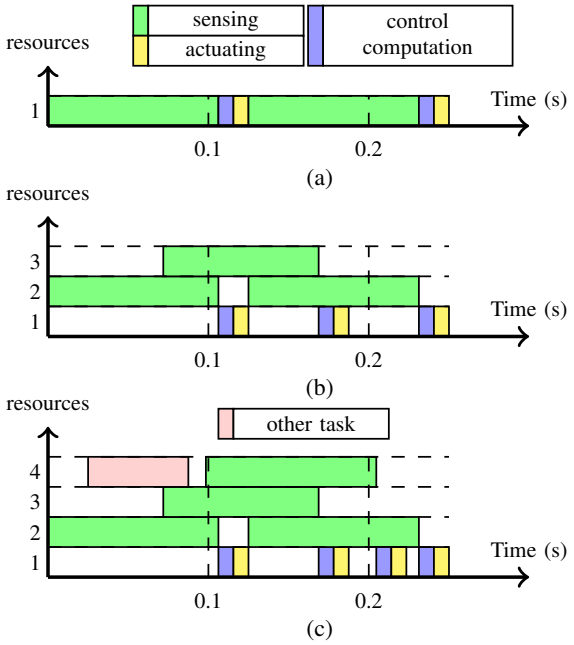


Fig. 1: Examples of resources configurations. a) sequential b) Pipelined with 3 cores. c) Reconfigured pipeline: the cores (4 processing units) are dynamically assigned to the sensing pipeline or to other tasks.

Contributions: We present a state-based modelling strategy for pipelined sensing controllers that is applicable to reconfigurable pipelined sensing. This modelling strategy is used to design reconfigurable controllers that outperform the static pipelined implementation, while guaranteeing stability.

Organization: Section II starts with related work. Section III presents the modelling strategy for control systems with pipelined sensing and reconfigurable pipelined sensing.

Theoretical background and a control design strategy for reconfigurable pipelined sensing are proposed in Section IV. Section V introduces a case study where reconfigurability improves the quality of control. Section VI presents the results for validation of the proposed strategy. Conclusions and future work are discussed in Section VII.

II. Related work

Strategies for coping with a long sample period are found in the embedded domain and in the control domain. Examples of control approaches can be found in [8]–[12]. Estimation techniques based on Kalman filters are used in [8], [9]. Approaches in [10], [11] allow an actuation period shorter than the total latency by using multi-rate strategies. In the embedded domain, the approaches consist of reducing the image processing latency by creating faster parallel implementations of the algorithms in specialized hardware such as GPUs [13], [14] or FPGAs [15], [16]. However, these strategies for coping with a long sample period have some limitations. Control strategies rely on the system model to estimate sensing information. Such estimations are vulnerable to modelling errors, they are unable to predict disturbances, and their prediction errors increase with longer delays. Embedded strategies require a great effort to speed up the image processing algorithm and the final latency may not be shorter than the recommended sample period of the controller.

An alternative to cope with long sample delays is to combine knowledge from both control and embedded domains to pipeline the sensing algorithm. Pipelined sensing control was introduced by Krautgartner and Vincze in [5] and extended in [6], [7]. They compared the performance of sequential and pipelined sensing in visual servo-positioning systems by using multiple control strategies. The pipelined sensing is shown to outperform the sequential sensing in all tested controllers. These results were only applied to visual servo-positioning systems; they did not introduce a modelling strategy based on the state space for pipelined sensing control, and they did not investigate the possibility of reconfiguring the sensing pipeline. Our work considers these aspects. We introduce a modelling strategy for pipelined systems which is applicable to a wide class of linear systems and we cope with the reconfigurability by extending the model and presenting a control design strategy.

III. Modelling Control Applications

Notation preliminaries on resources: Consider a multi-core platform with N cores. Such platform executes a set A of M sporadic applications along with a control application. The controller is composed by three tasks: T_{ac} acquisition, T_s sensing, T_c control computation, and T_a actuation. T_{ac} is performed by a camera. T_s, T_c, and T_a are executed by the resources. One of the N cores periodically executes the control computation T_c and actuator T_a tasks. The remaining cores are shared by T_s and the set A of sporadic applications.

Notation preliminaries on control: linear time-invariant control applications of the following form are considered:

$$\begin{aligned} \dot{x}(t) &= Gx(t) + Hu(t-\tau) \\ y(t) &= Cx(t) \end{aligned} \tag{1}$$

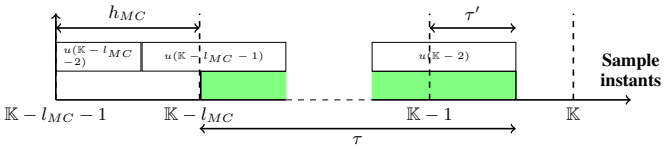


Fig. 2: Relationship between the delays τ, τ', and l_{MC}, and the sample period h_{MC}.

where $G \in R^p \times R^p$ is the state matrix, $H \in R^p \times R^q$ the input matrix, $C \in R^r \times R^p$ the output matrix, $u(t-\tau) \in R^q$ the controller output vector, $x(t) \in R^p$ the state vector, $y(t) \in R^r$ the output vector, and τ the sensor-to-actuator delay given by:

$$\tau = \tau_{ac} + \tau_s + \tau_c + \tau_a$$

with τ_{ac}, τ_s, τ_c, and τ_a the acquisition, sensing, control, and actuating latencies respectively and $\tau_s >> \tau_a + \tau_{ac} + \tau_c$

Our Reconfigurable Pipelined Controller (RPC) is composed by a Maximal Configuration (MC) and n Reduced Configurations (RCs) $\{RC1, RC2, ..., RCn\}$. A MC uses the maximum amount of resources designated for the IBC N_{MC} while the RCs use a smaller amount of resources $N_{RC} < N_{MC}$ to release cores for other applications. A RPC is active when the system is capable of switching between a MC and one or more RCs. For example, Fig. 1c shows a RPC switching between a MC with N_{MC}=4 and a RC with $N_{RC} = 3$.

A. Modelling MC

Consider an IBC with $N_{MC} < N$ processing units available for computation. Since T_c and T_a are executed in a fixed processing unit, only $N_{MC} - 1$ processing units are available for T_s. Eq. 1 is discretized by using a zero order hold and the sample period of a MC h_{MC} is:

$$h_{MC} = \frac{\tau}{N_{MC} - 1} \tag{2}$$

leading to a discrete state space representation of the form:

$$\begin{aligned} x(\mathbb{K}+1) = \Phi_d x(\mathbb{K}) + \Gamma_\alpha(\tau)u(\mathbb{K} - l_{MC}) + \\ \Gamma_\beta u(\mathbb{K} - l_{MC} - 1) \end{aligned} \tag{3}$$

with [17]:

$$\begin{aligned} \Phi_d &= e^{Gh_{MC}} \\ \Gamma_\alpha(\tau') &= \int_0^{h_{MC}-\tau'} e^{Gs} H ds \\ \Gamma_\beta(\tau') &= \int_{h_{MC}-\tau'}^{h_{MC}} e^{Gs} H ds \end{aligned} \tag{4}$$

where $\tau' = \tau - (l_{MC} - 1)h_{MC}$, $l_{MC} = \lceil \frac{\tau}{h_{MC}} \rceil$, $u(\mathbb{K} - l_{MC} - 1)$, and $u(\mathbb{K} - l_{MC})$ are the delayed controller outputs, $x(\mathbb{K})$ is the discrete time state vector. Fig. 2 illustrates the relationship between different timing components. The sample period h_{MC} should be chosen such that:

$$l_{MC} = N_{MC} - 1 \tag{5}$$

If $l_{MC} < (N_{MC} - 1)$ not all the available cores are used

for running the sensing algorithm, leading to a longer sample period. Redefining the states from Eq. 3 as

$$z(\mathbb{K}) = [x^T(\mathbb{K}),\, u^T(\mathbb{K}-l_{MC}),\, u^T(\mathbb{K}-l_{MC}+1)\cdots \\ u^T(\mathbb{K}-2),\, u^T(\mathbb{K}-1)]^T \quad (6)$$

the following augmented system is obtained:

$$z(\mathbb{K}+1) = \Phi_{MC} z(\mathbb{K}) + \Gamma_{MC} u(\mathbb{K}) \quad (7)$$

$$\Phi_{MC} = \begin{bmatrix} \Phi_d & \Gamma_\alpha(\tau') & \Gamma_\beta(\tau') & \dots & 0 \\ 0 & 0 & I & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & I \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \quad (8)$$

$$\Gamma_{MC} = \begin{bmatrix} 0 & 0 & \dots & 0 & I \end{bmatrix}^T \quad (9)$$

where $\Phi_{MC} \in R^{p+l_{MC}} \times R^{p+l_{MC}}$ and $\Gamma_{MC} \in R^{p+l_{MC}} \times R^{q}$ are the state and input matrix respectively.

Eqs. 6-9 correspond to a system under a MC. Since this model includes τ_s, h_{MC}, and a fixed number of sensing instances $N_{MC}-1$, it can be used for modelling control applications with static pipelined sensing.

In the state definition of Eq. 6, the number of states depends on l_{MC} (i.e. the number of available cores). In a RC, l_{RC} increases with the number of available cores for sensing (See Eq. 12) leading to a discrete time model with fewer states than the MC. However, in order to design a RPC, it is necessary that the discrete time models of the MC and the RC have an equal number of states. In the next section, we propose a modelling strategy for the RCs that addresses this aspect.

B. Modelling RC

Consider a reduced pipelined sensing controller with N_{RC} processing units available for computation. The sample period of the RCs is defined by:

$$h_{RC} = \frac{\tau}{N_{RC}-1} \quad (10)$$

Using h_{RC} and Eqs. 3 and 4 the continuous time system in Eq. 1 is discretized. The new state definition is given by:

$$z(\mathbb{K}) = [x^T(\mathbb{K}),\, u^T(\mathbb{K}-l_{RC}),\, u^T(\mathbb{K}-l_{RC}+1)\cdots \\ u^T(\mathbb{K}-2),\, u^T(\mathbb{K}-1)]^T \quad (11)$$

with

$$l_{RC} = N_{RC} - 1 \quad (12)$$

Since $N_{RC} < N_{MC}$, $l_{RC} < l_{MC}$. Therefore the state vector in Eq. 6 has more states than the state vector in Eq. 11. In order to equalize the number of states in all configurations, Eq. 11 is augmented by introducing old controller outputs u in the state vector definition, so that the length of $z(\mathbb{K})$ in the RC is equal to its length in the MC:

$$z(\mathbb{K}) = [x^T(\mathbb{K}), u^T(\mathbb{K}-l_{MC}), u^T(\mathbb{K}-l_{MC}+1), \cdots, \\ u^T(\mathbb{K}-l_{RC}-1), u^T(\mathbb{K}-l_{RC}), u^T(\mathbb{K}-l_{RC}+1), \cdots, \\ u^T(\mathbb{K}-3), u^T(\mathbb{K}-2), u^T(\mathbb{K}-1)] \quad (13)$$

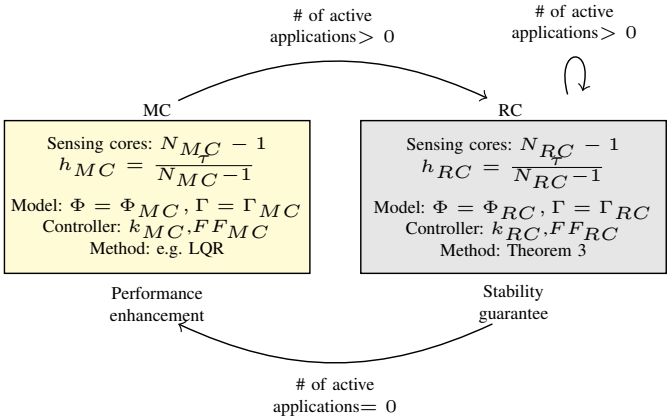


Fig. 3: Operation modes of a RPC. One of the RCs becomes active when one or multiple tasks from the set A are triggered and executed on the resources. Otherwise the MC becomes active.

where $x^T(\mathbb{K})$ and $u^T(\mathbb{K})$ are the discrete time state vector and controller output of the RC respectively. Therefore, the augmented discrete state space matrices are denoted by:

$$z(\mathbb{K}+1) = \Phi_{RC} z(\mathbb{K}) + \Gamma_{RC} u(\mathbb{K}) \quad (14)$$

$$\Phi_{RC} = \begin{bmatrix} \Phi_d & 0 & \dots & \overbrace{\Gamma_\alpha(\tau')}^{l_{RC}} & \Gamma_\beta(\tau') & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & I & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 0 & 0 & \dots & I \\ 0 & \underbrace{0}_{l_{MC}} & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \quad (15)$$

$$\Gamma_{RC} = \begin{bmatrix} 0 & 0 & \dots & 0 & I \end{bmatrix}^T \quad (16)$$

where $\Phi_{RC} \in R^{p+l_{MC}} \times R^{p+l_{MC}}$ and $\Gamma_{RC} \in R^{p+l_{MC}} \times R^{q}$ form the augmented state input matrix for an RC.

The model of the MC (Eqs. 6-9) and of the RCs (Eqs. 13-16) are used to design control systems in the next section.

IV. Control design strategy

Depending on the number of available cores, the RPC switches between a MC and one or more RCs, which can potentially lead to an unstable closed loop behaviour [18]. The control design strategy has to cope with this problem. The purpose of this section is:

- to design feedback gains k_i which allow stable switching between the above mentioned sensing configurations, as is shown in Fig. 3;

- to design a feed-forward gain FF_i such that the output vector y converges to the input step reference R. This problem is called set-point regulation.

These results provide a RPC of the following form:

$$u(\mathbb{K}) = k_i z(\mathbb{K}) + FF_i R \tag{17}$$

where the sub-index i denotes one of the sensing configurations $i = \{MC, RC\}$.

In the following subsection we provide an overview for our strategy to design a RPC. Theoretical background and the detailed controller design strategy are provided in Section IV-B and Section IV-C respectively.

A. Design summary

Given a sensor-to-actuator delay τ, a continuous time model of the form of Eq. 1, and resources running a set of sporadic applications A, a RPC with maximum $N_{MC} - 1$ sensing units is designed following these design steps:

1) **Modelling MC**. Find sample period h_{MC} using Eq. 2. Discretize the continuous model with Eq. 4. Construct the discrete time model using Eqs. 6-9.
2) **Controller design MC**. Find a feedback gain k_{MC} using Linear Quadratic Regulator (LQR) or pole-placement such that the performance (e.g. settling time, quadratic cost) is enhanced. Find a feed-forward gain FF_{MC} from Theorem 3.
3) **Modelling RCs**. Based on the M sporadic tasks, decide how many RCs n are required; this is a design parameter. For each configuration, determine the available number of cores N_{RC} and find the sample period h_{RC} by Eq. 10. Discretize the model for all the sample periods by Eq. 4 and construct the set of discrete time models following Eqs. 13-16.
4) **Controller design RCs**. Find the feedback gains k_{RC} by solving the Linear Matrix Inequalities (LMIs) stated in Theorem 4. Design a feed-forward gain FF_{RC} for each RC by using Theorem 3.

Following these design steps, a RPC is designed. Such a controller is capable of stable switching between a MC and one or more RCs. The next subsections introduce detailed information about the controller design.

B. Theoretical background

Consider the discrete time models from Eqs. 7 and 14 with control law $u(\mathbb{K}) = k_i z(\mathbb{K})$. The closed loop representation is given by:

$$z(\mathbb{K}+1) = (\Phi_i + \Gamma_i k_i)\, z(\mathbb{K}) \tag{18}$$

The stability requirements of the systems in Eq. 18 are defined in Theorem 1 for an individual system, and in Theorem 2 for the set of systems. Theorem 3 defines the feedforward gain FF_i introduced in Eq. 17.

Theorem 1. *A discrete time model $\Phi_i + \Gamma_i k_i$ is stable if and only if there exist positive definitive matrices $P \succ 0$ and $Q_i \succ 0$ that satisfy the Discrete Time Lyapunov Equation (DTLE) $P - (\Phi_i + \Gamma_i k_i)^T P\,(\Phi_i + \Gamma_i k_i) \succ Q_i$ [19].*

Theorem 2. *Given a set of models of the form $\Phi_i + \Gamma_i k_i$, $V(z) = z^T P z$ is a Common Quadratic Lyapunov Function (CQLF) if there exist positive definitive matrices $P \succ 0$ and $Q_i \succ 0$ such that P is the simultaneous solution of the DTLEs*

$$P - (\Phi_i + \Gamma_i k_i)^T P\,(\Phi_i + \Gamma_i k_i) \succ Q_i.$$

The existence of a CQLF is a necessary and sufficient condition for stable arbitrary switching between the set of systems in Eq. 18. [20].

Theorem 3. *Given a control law of the form $u(\mathbb{K}) = k_i z(\mathbb{K}) + FF_i R$, the system states $z(\mathbb{K})$ converge to a desired reference R if FF_i is defined by [21]*

$$FF_i = \begin{bmatrix} k_i & I \end{bmatrix} \begin{bmatrix} \Phi_i - I & \Gamma_i \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix}$$

C. Controller design strategy

The control design is accomplished in two stages: (i) for the MC (ii) for n RCs.

1) gains in MC: As already explained in Section III-A, the MC has the largest number of cores in the RPC N_{MC}. Therefore, the MC has the shortest sample period in the RPC (see Eq. 2) and hence, the best control performance can potentially be achieved. To this end, a feedback gain k_{MC} is designed to enhance performance by using LQR or pole-placement [21]. Finally, a feed-forward gain FF_{MC} is computed using Theorem 3. The controller output for the MC is then given by:

$$u(\mathbb{K}) = k_{MC} z(\mathbb{K}) + FF_{MC} R$$

The above control law provides the desired performance as long as the system is running under the MC. However, due to the triggering of one or multiple sporadic tasks of the set A, the system switches to the RCs as illustrated in Fig. 3.

2) gains in RCs: The controllers for the RCs are designed to guarantee stability if *arbitrary* switching occurs between the MC and the n RCs. To this end, the feedback gains k_{RC} are found by solving the LMIs from Theorem 4. Finally, a feed-forward gain FF_{RC} for each RC is found using Theorem 3.

Theorem 4. *Consider the models of Eq. 18. If there exist positive definitive matrices $O \succ 0$ and $Q_{MC} \succ 0$, $Q_{RC} \succ 0$, Y_{MC} and Y_{MC} such that the following LMIs hold*

$$\begin{pmatrix} O & (\Phi_{MC} O + \Gamma_{MC} k_{MC} O)^T & O \\ (\Phi_{MC} O + \Gamma_{MC} k_{MC} O) & O & 0 \\ O & 0 & Q_{MC}^{-1} \end{pmatrix} \succ 0 \tag{19}$$

$$\begin{pmatrix} O & (\Phi_{RC} O + \Gamma_{RC} Y_{RC})^T & O \\ (\Phi_{RC} O + \Gamma_{RC} Y_{RC}) & O & 0 \\ O & 0 & Q_{RC}^{-1} \end{pmatrix} \succ 0 \tag{20}$$

then the models in Eq. 18 have a CQLF with a feedback gain $k_{RC} = Y_{RC} O^{-1}$.

Proof: Applying the Schur complement on Eq. 20 yields:

$$O - (\Phi_{RC}O + \Gamma_{RC}Y_{RC})^T O^{-1} (\Phi_{RC}O + \Gamma_{RC}Y_{RC}) - OQ_{RC}O \succ 0$$

Defining $O = P^{-1}$ and $Y_{RC} = k_{RC}P^{-1}$:

$$P^{-1} - (\Phi_{RC}P^{-1} + \Gamma_{RC}k_{RC}P^{-1})^T P(\Phi_{RC}P^{-1} + \Gamma_{RC}k_{RC}P^{-1}) - P^{-1}Q_{RC}P^{-1} \succ 0$$

Pre and post multiplying by P:

$$PP^{-1}P - P(\Phi_{RC}P^{-1} + \Gamma_{RC}k_{RC}P^{-1})^T P$$
$$(\Phi_{RC}P^{-1} + \Gamma_{RC}k_{RC}P^{-1})P - PP^{-1}Q_{RC}P^{-1}P \succ 0$$

$$P - (\Phi_{RC} + \Gamma_{RC}k_{RC})^T P (\Phi_{RC} + \Gamma_{RC}k_{RC}) \succ Q_{RC} \quad (21)$$

An identical procedure is applied to the LMIs in Eq. 19:

$$P - (\Phi_{MC} + \Gamma_{MC}k_{MC})^T P (\Phi_{MC} + \Gamma_{MC}k_{MC}) \succ Q_{MC} \quad (22)$$

From Eqs. 21 and 22 it is clear that the systems defined by Eq. 18 have a CQLF defined in Theorem 2. This completes the proof. ■

The controller output for the RC is then defined by:

$$u(\mathbb{K}) = k_{RC}z(\mathbb{K}) + FF_{RC}R$$

This section presents a control design strategy for a RPC. This controller enhances performance if a MC is used and stability with temporarily degraded performance if the RCs are used. The next section provides a motivational set-up that shows how RPC is beneficial to control performance.

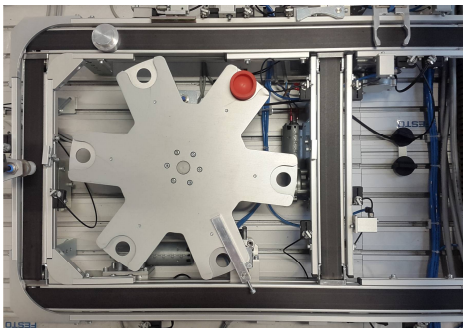
V. Case Study: xCPS

We evaluate our approach through simulations based on an experimental platform eXplore Cyber-Physical Systems (xCPS) called [22]. An Image-Based Controller and multiple applications share a multi-core platform in the system. The applications are sporadically triggered allowing their cores to be temporarily reused by the controller in order to enhance the controller performance.

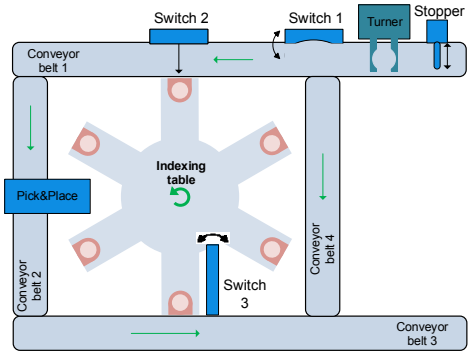
The xCPS platform is an industrial assembly line simulator which is used for teaching and research purposes. The machine assembles or disassembles circular complementary objects that come in two shapes: lower and upper parts. The assembly section of xCPS (Fig. 4a) is considered in our case study. The following subsections describes detailed information about the xCPS assembly process.

A. Assembly tasks and resources

A schematic of the assembly hardware is depicted in Fig. 4b. Four tasks are involved in the assembly process: A Supervisory Control Task (SCT), a Turner Task (TT), an Indexing Table Task (ITT), and an Image-Based Control (IBC). The assembly process is controlled by means of the SCT, which is activated whenever a new assemble object enters *conveyor belt 1*. The SCT also generates set-points for the controllers, activates the system actuators (such as *switches*, the *stopper* and the *pick and place unit*), and distributes the available resources among the assembly tasks. The TT corrects



(a)



(b)

Fig. 4: Assembly section on xCPS. a) camera view. b) schematic.

the orientation of the assembly objects in case it is necessary, by manipulating the *turner* actuator. The ITT consists of a local controller which aligns an *indexing table* either with *switch 2* or with a *pick and place* actuator. The IBC regulates the speeds of the conveyor belts using DC motors. A camera and an image processing algorithm are used as sensor to measure the speed of the belts. The speed is kept low when an object is going through the actuators (e.g. turner, switch 2, and pick and place) and high otherwise. All three applications (SCT, ITT, and TT) are triggered sporadically based on the arrival of a new object whereas the controller always remains active.

A 6 core multiprocessor board ($N = 6$) is considered as multi-core platform to simulate the assembly tasks. The three mentioned sporadic applications ($M = 3$ with $A = \{SCT, TT, SCT\}$) and the controller are executed on these 6 cores. Each task runs in a separate core in case they need to be executed at the same time.

B. Assembly process

The assembly process begins when an object is fed into *conveyor belt 1* and it is blocked by a *stopper*. The SCT is immediately triggered to generate controller references and activation times of the actuators which guarantee the correct assembly of the object. The SCT retracts the *Stopper* and the object moves through a *turner* activating the TT. Lower objects go to an *indexing table* using *switch 2* and trigger the ITT, whereas upper objects go to *conveyor belt 2*. A *pick and place* actuator grabs the upper objects from *conveyor belt 2* and push them to the lower parts in the *indexing table*. Assembled

objects are moved by *switch 3* into *conveyor belt 3*. The system also rejects objects by using *conveyor belt 4* and *switch 1*.

Since the conveyor belts speeds are regulated by the IBC, the time elapsed between an object leaving the *stopper* and reaching the *turner* or the *indexing table* is known. Therefore, the activation time of the ITT and TT can be predicted. On the other hand, the SCT is triggered by the arrival of a new object; therefore its activation time can not be predicted. This is particularly important for the design of the reconfigurable controller in the next section. The throughput of the machine is directly affected by the Settling time (S) of the control system: a shorter S means that the blocks reach each stage of the assembly process faster, increasing the number of assembled blocks per time unit. So, optimizing the performance of the controller increases the throughput of the system.

VI. Simulation results

In this section four different sensing configurations of the Image-Based Controller are studied and their performance is evaluated on xCPS. The settling time (S) of the control system is used as performance metric since it has a direct impact on the throughput of xCPS.

A. Image-Based Controller design

The IBC manipulates the speed of the conveyor belts on xCPS by adjusting the input voltage to the DC motors. The parameters of the motor model in Eq. 1 are $G = -49.5050$, $H = 0.01$, and $C = 1$. The reference R is the desired speed of the DC motor. The controller is designed to reach such a reference as fast as possible (i.e., shortest S). The sensor-to-actuator delay is $\tau = 0.125s$. Given the number of sporadic tasks (SCT, TT, ITT with $M = 3$) and the multi-core platform ($N = 6$), multiple IBCs are designed by changing the number of cores (N_{MC}) assigned to the controller:

1) Serial configuration: A serial configuration (denoted by SC) uses the same core for executing the T_s, T_c, and T_a tasks as shown in Fig. 5a. The sensor-to-actuator delay $\tau = 0.125s$ implies a sample period $h = \tau = 0.125s$. Since the sample period is known, Eq. 4 is used to discretize the continuous time model. The states are redefined using Eqs. 6-9 to include the delay in the state space notation. An IBC is designed by using standard a pole-placement technique [21]. The poles are tuned such that the shortest settling time is achieved. The resulting feedback gain is

$$k_{SC} = [-17 \times 10^{-9} \ \ -16 \times 10^{-6}].$$

Theorem 3 is used to find the feed-forward gain $FF = 0.5$.

2) Pipelined configuration: Using the two free cores in the board, an IBC with two sensing pipes (denoted by PC) is designed, as illustrated in Fig. 5b. The control and actuation tasks are executed on a different core in order to locally maintain the required information for computing the controller output such as FF_{PC}, k_{PC}, and old $u(\mathbb{K})$. Since the number of available cores for sensing is constant, the controller is designed using the guidelines of Section IV-A for a MC, considering that $N_{MC} = 3$. The resulting sample period is $h_{MC} = 62.5 \times 10^{-3}s$. The feed-forward gain is given by

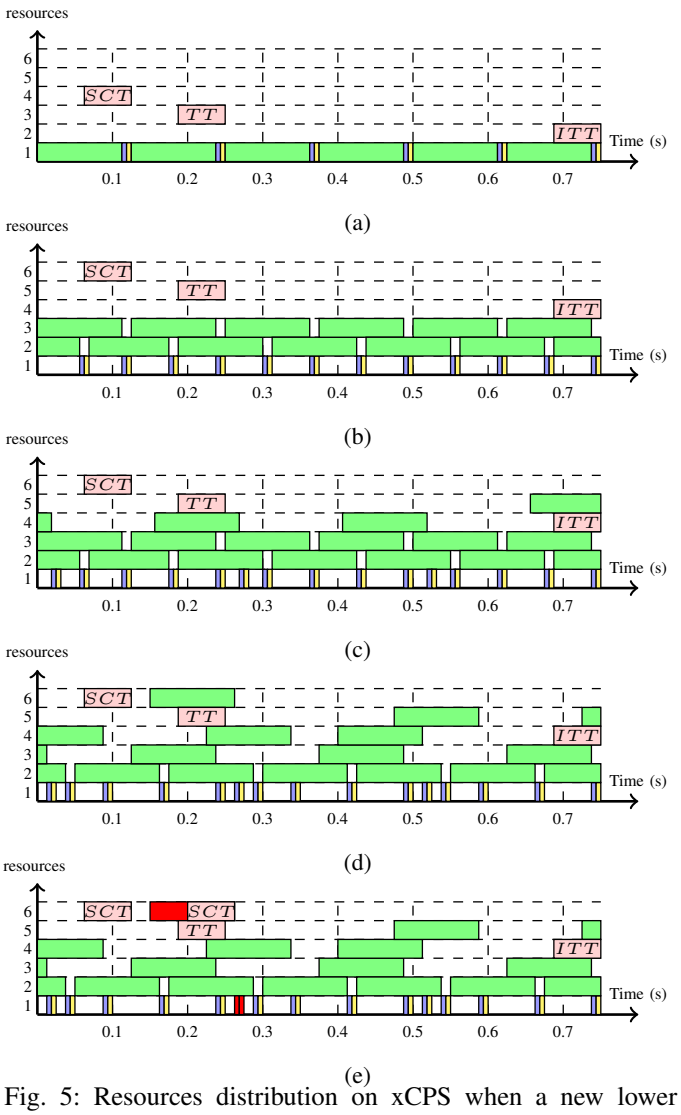


Fig. 5: Resources distribution on xCPS when a new lower part object arrives. The colours follow those of Fig. 1. a) SC b) PC c) RPC with $N_{MC} = 5$ d) RPC with $N_{MC} = 6$ and no interruption from SCT e) RPC with $N_{MC} = 6$ with interruption from SCT at time 0.2s. No sensing information is delivered at time 0.275s (see red squares).

$FF_{MC} = 0.5$. Similar to the SC, a pole-placement controller is tuned to achieve the shortest S giving the feedback gain

$$k_{PC} = [-0.04 \ \ -1.99 \ \ -44.02] \times 10^{-3}.$$

3) Five core RPC: Fig. 5b shows that the ITT and TT are applications with a low frequency of appearance. Such applications are triggered by the SCT, making it possible to schedule them in such a way that their cores are temporarily reused by the sensing instances of a RPC. This scenario is shown in Fig. 5c. The procedure presented in Section IV-A is used to design the control system. The RPC is composed of two sensing configurations: a MC with 4 sensing cores ($N_{MC} = 5$) and one RC (n=1) $\{RC1\}$ with 2 sensing cores ($N_{RC1} = 3$). The MC has a sample period $h_{MC} = 31.25 \times 10^{-3}s$. The feed-forward gain is $FF_{MC} = 0.61$. Using the procedure described for the sequential controller, a pole-placement controller is

tuned to find the feedback gain resulting in

$$k_{MC} = [0.2 \ \ -1.6 \ \ -7.8 \ \ \ -37 \ \ -174] \times 10^{-3}.$$

The sample period of the $RC1$ is $h_{RC1} = 62.5 \times 10^{-3}s$. The feed-forward gain is $FF_{RC1} = 0.3138$. The set of LMIs [23] from step 4 in the design summary are solved with the optimization toolbox YALMIP [24] and a semi-definite quadratic solver SDPT3 [25]. The resulting feedback gain is

$$k_{RC} = [0.3 \ \ 2.9 \ \ 13.2 \ \ 62.2 \ \ 293] \times 10^{-3}.$$

4) Six core RPC: In order to reuse all resources of the sporadic tasks, the procedure from Section IV-A is used for designing a RPC which uses 6 cores. Since the activation time of the SCT is not predictable, two cases are considered: when no interruption occurs in the sensing pipeline (e.g. SCT triggers when its sensing core is available as shown in Fig. 5d) and when it does (e.g. SCT preempts the sensing task as shown in Fig. 5e). The RPC is composed by three sensing configurations: a MC with 5 pipes ($N_{MC} = 6$), and two RC ($n = 2$) $\{RC1, RC2\}$. $RC1$ uses two pipes ($N_{RC1} = 3$), and $RC2$ uses three pipes ($N_{RC2} = 4$). For the MC the parameters are $h_{MC} = 25 \times 10^{-3}s$, $FF_{MC} = 0.68$, and

$$k_{MC} = [-0.3 \ \ -1.9 \ \ -6.5 \ \ -22.6 \ \ -78 \ \ -269] \times 10^{-3}.$$

For the $RC1$ the parameters are $h_{RC1} = 62.5 \times 10^{-3}s$, $FF_{RC1} = 0.28$, and

$$k_{RC1} = [0.4 \ \ 2.4 \ \ 7.4 \ \ 25.7 \ \ 88.7 \ \ 306] \times 10^{-3}.$$

For the $RC2$ the parameters are $h_{RC2} = 50 \times 10^{-3}s$, $FF_{RC2} = 0.29$, and

$$k_{RC2} = [0.4 \ \ 2.2 \ \ 6.9 \ \ 23.8 \ \ 82 \ \ 284] \times 10^{-3}.$$

B. Controllers simulations

The performance of the controllers designed in the previous subsection is computed by simulating the arrival of a new lower part object on the conveyor belt. For simplicity, only the speed on the first conveyor belt is illustrated in the figures. The reference is initially kept low while the object passes through the *turner*. When the TT is finished, the speed is temporarily increased to reach the next stage of the assembly process. The object then moves to the *indexing table* activating the ITT. The tasks distribution on the multi-core platform is shown in Fig. 5 for the different sensing configurations. In absence of disturbances, sensing errors, and modelling errors, zero steady state error is achieved in the cases.

1) SC vs PC: The model output and the controller output are plotted in Fig. 6. It is clear that the controller with pipelined sensing (red line) outperforms the serial controller (purple line). The controller in SC actuates once every sensor-to-actuator delay τ, whereas the controller in PC actuates twice in the same period. As a consequence, a more aggressive controller output is achieved by the PC controller. At time 0.25s, the controller output of the PC controller is initially higher than the SC controller but it switches to the same value in the following sample. The PC controller (red line) achieves a settling time 29% shorter than the SC controller (purple). It can be seen in Fig. 6 that as a result of the sensing latency τ_s, a change on the controller output is not immediately seen in the motor speed.

TABLE I: Performance comparison of sensing configurations.

Sensing configuration	$S\ ms$	Performance enhancement %
SC	77	0
PC	55	29
RPC with $N_{MC} = 5$	40	27
RPC with $N_{MC} = 6$	24	38

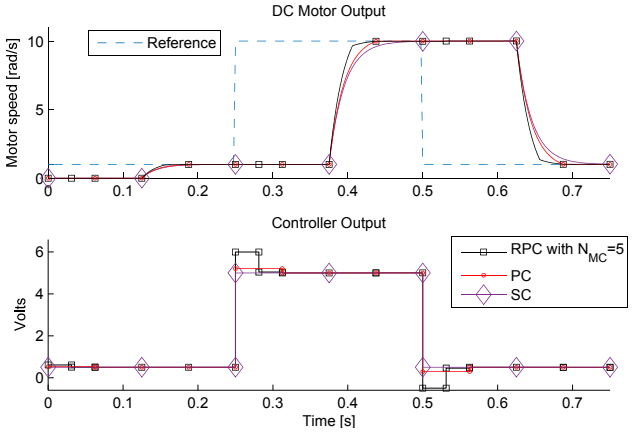


Fig. 6: Comparison: SC, PC, and RPC with $N_{MC} = 5$. Markers denotes the time when the controller is actuating. The switching sequence of RPC is $\{MC, MC, RC2, RC1, RC1\}$.

2) PC vs five core RPC: The model output and controller output are plotted in Fig. 6. The performance of the RPC controller is enhanced by using MC when the system is away from the reference. $RC1$ is used when the system is on the reference. As a result, RPC actuates more frequently when the reference changes. The RPC controller outperforms the PC controller by reaching the reference 27% faster in all set-point changes. This result is summarized in Table I.

3) Five core RPC vs six core RPC: If the arrival time of a new block is known in advance, the SCT can be scheduled in such a way that the RPC with $N_{MC} = 6$ uses the MC when there is a deviation from the reference and the RCs $RC1$ and $RC2$ are used when the system is on the reference. The simulation from Fig. 7 shows such a scenario. The RPC with $N_{MC} = 6$ (red line) reaches the reference 38% faster than the RPC with $N_{MC} = 5$ (black line). This result is summarized in Table I. However, if an unexpected block enters the system, the SCT is triggered and the sensing information might be dropped. In Fig. 7, the RPC with $N_{MC} = 6$ (purple line) is forced to switch from MC to $RC2$ because of the unexpected activation of SCT at time 0.2s. $RC2$ assures stability of the controller, at the cost of control performance.

C. Discussion

From the simulations described in this section, the following insights are deduced: first, pipelined control outperforms serial control due to the higher sample rate, at the cost of more resources used [5]. Second, a RPC enhances the control performance if the arrival times of the sporadic tasks are known. If not, the controller is still stable with a lower performance. Third, in case there is no predictability on one or multiple sporadic applications, it is recommended to not

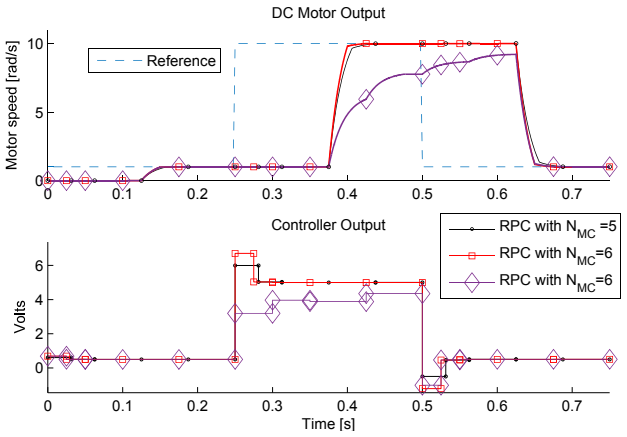


Fig. 7: Comparison between the RPCs with $N_{MC} = 5$ and $N_{MC} = 6$. The configuration with $N_{MC} = 6$ has a switching sequence of $\{MC, MC, RC2, RC1, RC1\}$ (red line). The sequence is changed to $\{MC, MC, RC2, RC1, RC1, RC2\}$ for the case of an unexpected block arrival (purple line).

include such cores in the RPC. Fourth, a larger number of resources in pipelined sensing controllers will most likely result in better control performance (See Table I).

VII. Conclusions and Future Work

We have presented reconfigurable pipelined control for Image-Based Controllers. A pipelined controller reduces the sample period of Image-Based Controllers by using additional sensing resources in a pipelined fashion. A reconfigurable pipelined controller reduces the sample period further by temporarily reusing the resources from other applications running on the same platform. We have introduced a state-based modelling strategy for pipelined and reconfigurable pipelined control, and a control design strategy for reconfigurable pipelined control. Simulation results show that pipelined control outperforms classical sequential implementations. Simulations also show that reconfigurable pipelined control outperforms pipelined control, if the maximal configuration is used when the output is deviated from the reference (e.g. in a transient state). Otherwise, reconfigurable pipelined control is still stable, at the cost of certain performance degradation. Our simulation results suggest that increasing the number of cores in pipelined sensing implementations leads to an improvement of performance. However, this needs further exploration over a wider range of systems. Further work may include the modelling of reconfiguration time, sensing errors, variable sensing latency, uncertainties, and disturbances. Considering Image-Based Control with parallelized and pipelined sensing simultaneously is also interesting. This provides further opportunities to reduce the sample period formulating a more challenging reconfiguration and resource allocation problem.

Acknowledgment

This work was funded by the Dutch Technology Foundation STW as part of the Robust Cyber-Physical Systems (rCPS) program, project 12697.

References

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

[2] K. Jo *et al.*, "Development of autonomous car—part II: A case study on the implementation of an autonomous driving system based on distributed architecture," *Industrial Electronics, IEEE Trans. on*, vol. 62, no. 8, pp. 5119–5132, 2015.

[3] F. Chaumette and S. Hutchinson, "Visual servo control. I. Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, pp. 82–90, 2006.

[4] P. Sharkey and D. Murray, "Delays versus performance of visually guided systems," *Control Theory and Applications, IEE Proceedings*, vol. 143, no. 5, pp. 436–447, 1996.

[5] P. Krautgartner and M. Vincze, "Performance evaluation of vision-based control tasks," in *Proc. ICRA'98*, vol. 3. IEEE, pp. 2315–2320.

[6] P. Krautgartner and M. Vincze, "Optimal image processing architecture for active vision systems," in *Computer Vision Systems*, ser. Lecture Notes in Computer Science. Springer, 1999, vol. 1542, pp. 331–347.

[7] S. Chroust *et al.*, "Evaluation of processing architecture and control law on the performance of vision-based control systems," in *Proc. AMC'00*. IEEE, pp. 19–24.

[8] H. Fujimoto, "Visual Servoing of 6 DOF Manipulator by Multirate Control with Depth Identification," in *Proc. CDC'03*, vol. 5. IEEE, pp. 5408–5413.

[9] K. Ito *et al.*, "Fast and accurate vision-based positioning control employing multi-rate kalman filter," in *Proc. IECON'13*. IEEE, pp. 6466–6471.

[10] A. Kawamura *et al.*, "Robust visual servoing for object manipulation with large time-delays of visual information," in *Proc. IROS'12*. IEEE, pp. 4797–4803.

[11] 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*, vol. 61, no. 2, pp. 1133–1143, 2014.

[12] C. Wang *et al.*, "Statistical learning algorithms to compensate slow visual feedback for industrial robots," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 3, 2015.

[13] R. Agrawal *et al.*, "A GPU based Real-Time CUDA implementation for obtaining Visual Saliency," in *Proc. ICVGIP'14*. ACM.

[14] van den Braak *et al.*, "Fast hough transform on GPUs: Exploration of algorithm trade-offs," in *Advanced Concepts for Intelligent Vision Systems*. Springer, 2011, vol. 6915, pp. 611–622.

[15] S. Kestur *et al.*, "Emulating mammalian vision on reconfigurable hardware," in *Proc. FCCM'12*. IEEE, pp. 141–148.

[16] L. Yao *et al.*, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *Proc. FPT'09*. IEEE, pp. 30–37.

[17] A. M. Annaswamy *et al.*, "Arbitrated network control systems: A co-design of control and platform for cyber-physical systems," in *Control of Cyber-Physical Systems*. Springer, 2013, pp. 339–356.

[18] Z. Sun and S. S. Ge, *Stability theory of switched dynamical systems*. Springer, 2011.

[19] W. J. Rugh, *Linear system theory*. Prentice hall Upper Saddle River, NJ, 1996, vol. 2.

[20] O. Mason and R. N. Shorten, "On common quadratic Lyapunov functions for stable discrete time LTI systems," *IMA Journal of Applied Mathematics*, vol. 69, pp. 271–283, 2002.

[21] J. L. Hellerstein *et al.*, *Feedback control of computing systems*. John Wiley & Sons, 2004.

[22] S. Adyanthaya *et al.*, "xCPS: A tool to eXplore Cyber Physical Systems," in *Proc. WESE'15*. ACM.

[23] S. Boyd *et al.*, *Linear matrix inequalities in system and control theory*. SIAM, 1994, vol. 15.

[24] J. Löfberg, "Yalmip : A toolbox for modeling and optimization in MATLAB," in *Proc. CACSD'04*. IEEE.

[25] R. H. Tütüncü *et al.*, "Solving semidefinite-quadratic-linear programs using SDPT3," *Mathematical programming*, vol. 95, pp. 189–217, 2003.