xCPS: A tool to eXplore Cyber Physical Systems *

Shreya Adyanthaya*, Hadi Alizadeh Ara*, João Bastos*, Amir Behrouzian*, Róbinson Medina Sánchez*, Joost van Pinxten*, Bram van der Sanden*, Umar Waqas*, Twan Basten*†, Henk Corporaal*, Raymond Frijns*, Marc Geilen*, Dip Goswami*, Martijn Hendriks†, Sander Stuijk*, Michel Reniers*, Jeroen Voeten*†

*Eindhoven University of Technology, Eindhoven, The Netherlands
†TNO ESI, Eindhoven, The Netherlands
http://www.xcps.info/

ABSTRACT

Cyber-Physical Systems (CPS) play an important role in the modern high-tech industry. Designing such systems is an especially challenging task due to the multi-disciplinary nature of these systems, and the range of abstraction levels involved. To facilitate hands-on experience with such systems, we develop a cyber-physical platform that aids in both research and education on CPS. This paper describes this platform, which contains all typical CPS components. The platform is used in various research and education projects for bachelor, master, and PhD students. We discuss the platform and illustrate its use with a number of projects and the educational opportunities they provide.

Keywords

Cyber-Physical Systems; embedded systems; education

1. INTRODUCTION

Cyber-Physical Systems (CPS) are becoming increasingly ubiquitous in society. CPS can be found in day-to-day situations such as traffic-light networks, smart homes, advanced automotive systems and smart energy systems, but also in almost any high-tech domain, such as medical imaging, electron microscopy, professional printing, and chip fabrication. These systems tightly integrate computation and communication and physical processes, where embedded computers and networks control those physical processes. Consequently, these systems pose several challenges due to their complexity and multidisciplinary requirements in all system development phases: design, analysis and implementation.

Designing a CPS is a challenging task, since exploring different design alternatives requires designers to understand and identify many different abstraction layers and very different domains and how they relate and interact. For instance, close to the physical-layer, local continuous feedback controllers are used to ensure correct functioning of actuators, while at a higher level, discrete supervisory controllers orchestrate the interactions among local controllers and globally control the entire system.

Even at the analysis stage of a design choice, predicting and evaluating performance and checking system requirements becomes highly complex because of interactions between different disciplines and system layers. Consider high-performance systems where control signals are required at very high frequencies. Therefore, computation and communication delays can no longer be neglected and must be taken into account to design the controllers.

Finally, looking at the implementation phase, new methodologies are needed that guarantee that the requirements are actually satisfied at all different layers of the system. Research in this area is focussed on automated synthesis techniques that provide guarantees (e.g. safety) and efficiency (e.g. maximal throughput) by construction.

All these challenges require not only research and innovation, but also a change of view in the domain and field, such that future engineers and researchers have a solid understanding as well as practical experience to address these challenges. Academic research and education play a crucial role by developing and disseminating methods and solutions to address the various issues described above.

In this paper we present eXplore Cyber-Physical Systems (xCPS), a platform of industrial complexity for research and education on CPS. It embeds a wide range of typical CPS components and is used as a vehicle for CPS education and research. It allows for experimentation, research and development of components relating to different disciplines, as well as multi-disciplinary aspects of a complex system. Furthermore, it gives students and researchers the chance to obtain a global view for all the steps in the development, from design [11] and analysis [24], to implementation [12] of CPS.

The focus of this paper is to demonstrate how the xCPS platform can help preparing students to face complex problems presents in today CPS industry. For this purpose, we illustrate the xCPS platform with current education projects being executed on it and show how these projects can improve learning and consolidation of acquired skills. Moreover, we describe current active research lines within the xCPS and provide examples that demonstrate their importance for CPS platforms.

Section 2 introduces the xCPS platform, Section 3 discusses education and research opportunities in different subjects using ongoing projects. Section 4 discusses the connections between the projects and the opportunities for integration and multidisciplinary approaches. Section 5 discusses re-

*This work is based on an earlier work: xCPS: A tool to eXplore Cyber Physical Systems, in *Workshop on Embedded and Cyber-Physical Systems Education, WESE*, © ACM, 2015. http://dx.doi.org/10.1145/2832920.2832923.

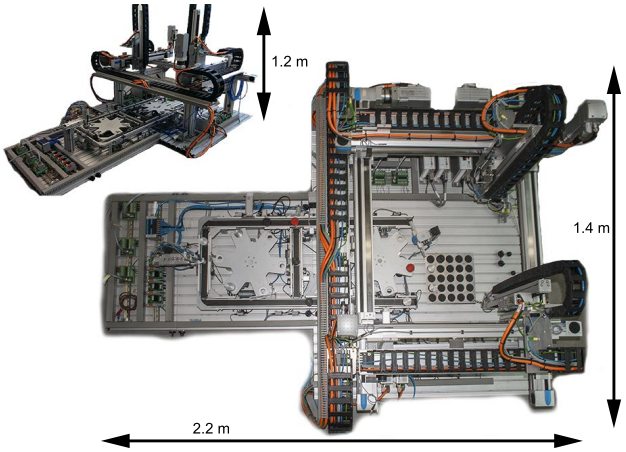


Figure 1: Top and side view of the xCPS platform.

lated work and Section 6 makes concluding remarks.

2. THE xCPS PLATFORM

The xCPS platform (Figure 1) is a small scale machine mimicking a production line that can assemble and disassemble objects. The cylindrical assembly pieces of xCPS (Figure 2) come in two complementary shapes, and three colours (silver, black or red). The cylindrical shape simplifies the transport and assembly process and different coloured pieces enable the creation of different jobs the platform is able to execute.

For reasons of cost and effort students cannot easily get access to perform experiments on actual industrial machines. Assumptions made by students, in the absence of measurements, cannot be validated and may be unrealistic for practical industrial platforms. Systems such as xCPS provide students and researchers with a realistic platform for measurements and analysis, enabling development of novel techniques that are closer and more applicable to practice.

2.1 Platform Overview

The system layout (Figure 3) illustrates the main components of the platform. The platform consists of one storage area, six conveyor belts, two indexing tables, two gantry arms, and several actuators and sensors. The storage area is a grid where 25 components can be stored.

There are six actuators called 'stoppers' in strategic positions along the assembly process that can obstruct the movement of pieces, effectively creating buffers accumulating pieces on the conveyor belts. Switches make it possible to change the route of individual objects. A turner can flip pieces. The platform also contains two actuators for assembly and disassembly of pieces. The pick & place actuator can clamp a part and combine it with a complementary part. A separator can disassemble two combined pieces. The xCPS platform is equipped with 15 sensors that can detect the presence of an object in the surrounding area. These sensors cannot distinguish the type of an object or its colour. To detect the type, colour, and location of an object, a camera can be added to the set-up.

There are many possible use cases of the xCPS platform.

Figure 2: Assembly pieces in the xCPS.

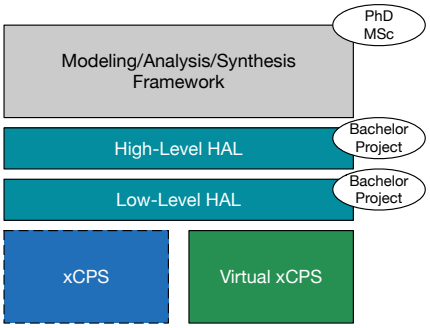


Figure 4: Architectural overview

One can for example simulate an assembly process where individual parts arrive on the first conveyor belt, are subsequently assembled and then put back into the storage. The gantry arms and storage area can be used as a sorting station, where the storage is sorted according to selected criteria, but it can also be used for games such as *tic-tac-toe* and *four-in-a-row*. Each of these use cases carry different objectives and learning goals.

2.2 Hardware Abstraction Layer

The mechanical hardware of the xCPS platform (i.e., belts, arms, picker) is controlled by electrical motors, servos and pneumatic actuators. These actuators are controlled by signals from several data acquisition and control input/output cards inside a general-purpose computer platform.

The Hardware Abstraction Layer (HAL) is essential to use the xCPS platform as an educational platform. This abstraction layer allows students to access and operate the physical system without detailed knowledge of the underlying realization. When desired, students can study the underlying layers and get hands-on experience in programming and controlling a real CPS.

The HAL for the xCPS platform is currently a C++-based Application Programming Interface (API) that consists of several functions that help the user program the system without requiring idetailed knowledge of the hardware layer. Figure 4 depicts the current architecture for xCPS. The highest abstraction includes the framework of behavioral models, schedulers and controller synthesis methods that are discussed in the next sections. This layer relies fully on the HAL to abstract from the underlying system details. For educational and research purposes, the HAL is implemented in two-levels. The low-level HAL consists of several functions that act as device drivers for the actuators and sensors. It is, for example, possible to control the gantry arms by sending

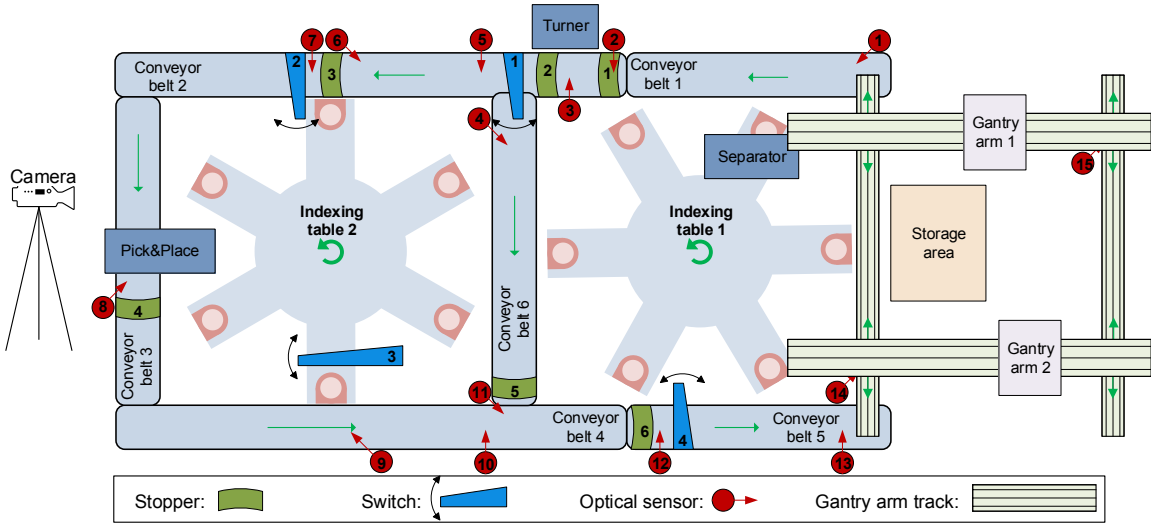


Figure 3: xCPS system layout.

them commands to move to specific 3-dimensional (x, y, z) locations. Such location commands are used by a high-level controller to hide the complex series of required physical signals beneath the HAL that operate the machine.

The high-level HAL for the HAL raises the abstraction so that the user does not need to concern him- or herself regarding the geometry and safety of the system. For instance, the user no longer needs to provide 3-dimensional locations in physical space, but instead can use function calls to move the gantry arms to logical locations for functional purposes, e.g., pick a piece, or go to homing positions. Furthermore, the API at this level also enforces certain minimal safety requirements, such that any series of command calls made by the user will not result in physical damage to the machine. Such damage could occur when trying to move to unreachable locations, or from colliding with other resources.

These projects allow students to get real experience developing and architecting software. Doing so, they develop skills related to both theoretical aspects and practical exercises. Theoretical aspects include functional correctness and safety with encapsulation, class organization, and API definition. Practical exercises include activities such as implementing the function calls of the API which need to comply with concurrent activations of actuators. In particular, the physical state of the machine needs to be maintained to ensure proper execution of a series of commands.

2.3 Virtualised platform

Especially during early development stages, it is often too expensive or time consuming to develop a physical prototype of each component in a system. The API introduced in Section 2.2 captures the higher-level behaviour of the mechanical hardware and can therefore be defined before a real prototype has even been developed. Such APIs are used to simulate the mechanical hardware (Simulator-in-the-Loop) for early development stages, to facilitate rapid prototyping and early design of control algorithms and software applications. On the other hand, Hardware-in-the-Loop simulations enable testing of individual hardware components in circumstances that are hard to reproduce in a real environment. Students can use simulations and visualizations when the real xCPS platform itself is not available or when it is not supporting certain functionality yet. Visualizations may show the physical environment, or focus on specific aspects of the activities, such as Gantt chart visualizations that focus on the ordering of activities in the system over time.

The xCPS machine has been modelled in the Blender 3D rendering software; renders of the model are shown in Fig 5. The actuators and mechanics that are essential for moving the pieces around are rendered. The physical interactions of the pieces and the actuators are modelled with the Blender physics simulation engine. The sensors are emulated by checking collisions with invisible regions. The simulation uses the same API as the real platform.

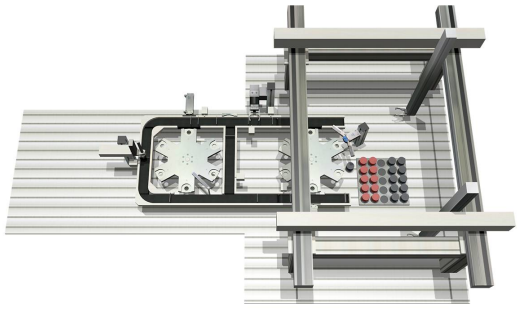
We have created a model in Parallel Object-Oriented Specification Language (POOSL) [26] that can simulate an environment (for instance non-deterministic or a predefined sequence of events) to test the robustness of controllers or software implementations for xCPS [14].

3. TEACHING AND RESEARCH OPPORTUNITIES IN THE xCPS PLATFORM

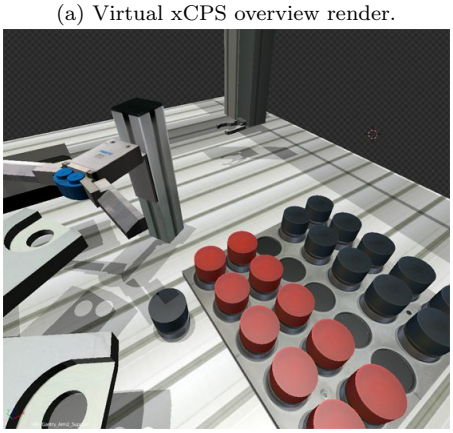
Exposing students and researchers to a platform such as xCPS gives them an opportunity to grasp the complexity of CPS. The overall layering of the system, from software-level to servo-level, creates an environment where multidisciplinary and cross-layer works are explored. Moreover, the different use cases for xCPS allow for a wide and diverse range of areas of expertise to be explored. Figure 6 depicts different development stages (Design, Analysis, Implementation), abstraction layers (System, Resources, Servo, Physical) and research areas that are investigated. In the next sections we address a number of these areas and explain why and how they can be used in education and research.

3.1 Example product flow

We consider an example *product flow* that uses a part of the machine to illustrate the techniques and learning outcomes. The product flow uses the following parts of the sys-



(a) Virtual xCPS overview render.



(b) Virtual xCPS detail render.

Figure 5: 3D visualization renders of the virtual xCPS in Blender.

tem (see Figures 3 and 9): bottom and top pieces are loaded by Gantry arm 1 onto conveyor belt 1. The piece is transported to the turner, which detects its position and places the right side up, The pieces continue onto conveyor belt 2, where switch 2 enforces whether they are loaded onto indexing table 2 or move further to conveyor belt 3. The pieces move towards the pick & place unit, where one top and one bottom are assembled into a single piece. The assembled pieces will travel on the indexing table until switch 3 forces it onto conveyor belt 4, where it finally gets unloaded after reaching the end of conveyor belt 5. We consider that either one top, or one bottom piece can be loaded at a time. Where necessary, the assumptions and abstractions made in the described product flow are refined to enable analysis capabilities of a particular technique.

3.2 Supervisory Control

Supervisory control coordinates and orchestrates actuation at the system level to achieve system goals (for instance that objects get assembled / disassembled). High-level control actions at supervisory control level are translated to fine-grained instructions sent to lower level controllers. The supervisor has to ensure that the system is free of deadlocks and eliminates any unsafe behaviour, such as gantry arms colliding.

High-tech CPS, such as the xCPS platform, typically consist of many sensors and actuators. Due to the number of components and the high level of interaction among them, proper modelling of the system and requirements is far from trivial. Requirements have to be at the right abstraction

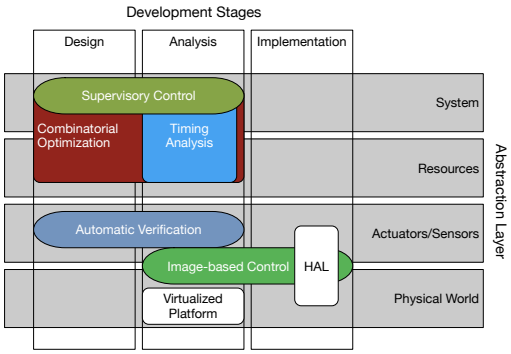


Figure 6: Positioning of the research areas that are explored using xCPS.

level, and the model should allow for local changes when the system configuration evolves. Due to the large state space, there are challenges in the synthesis step to automatically derive a supervisory controller that ensures satisfaction of the requirements.

In a bachelor level project [8] a student has modelled part of the platform to develop a supervisory controller following the synthesis-based model-based systems engineering process [4, 28]. In this process the uncontrolled system and the control requirements are modelled independently and in a modular way, using small, loosely coupled models based on the formal model of extended finite automata (EFAs) [23, 29]. The CIF 3 tool set [28] has been used to carry out the modelling.

A high-level, abstract, model has been used that partitions the conveyor belts into segments to simplify the control of the product flow. This abstraction is valid because of the stoppers that are present in xCPS, which enable a buffering mechanism. The student made the design decision that segments are allowed to hold at most one object. This simplifying restriction eases the reasoning about the needed requirements and the behaviour of the system for the student.

Examples of EFA models for a stopper (*Stopper1*), a sensor (*Sensor_Optical1*), and an area (*Area1*) are shown in Figure 7. Circles represent states and edges represent events that may change the state. Solid edges and dashed edges represent controllable and uncontrollable events [20, 21] in these automata, respectively. Controllable events are those which may be enforced or prevented by the supervisory controller; uncontrollable events cannot be prevented from occurring by a supervisory controller.

Requirements are formulated that specify correct operation of the actuators, and enforcing a correct product flow. The complete set of requirements (and requirement models) is presented in [8]. An example of such a requirement is that the number of items never exceeds the finite capacity of the areas on the conveyor belt and the indexing table. It can be formally expressed as an EFA. Figure 8 shows another example of a requirement model, expressed as an invariant propositional logic formula with propositions referring to properties of the machine state. The requirement states that the camera may only scan a work piece (notation $\rightarrow \{Camera1.on\}$ denotes the possibility to execute

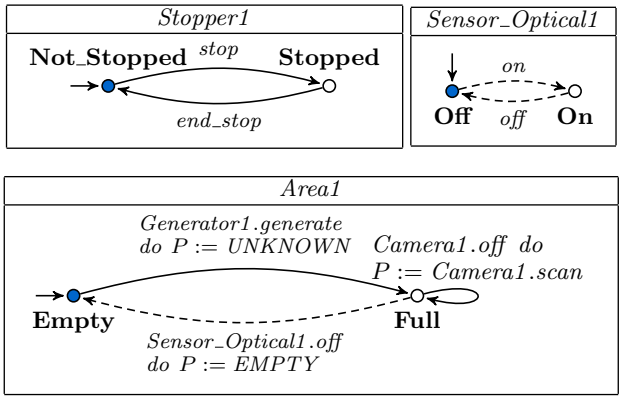


Figure 7: Example CIF 3 uncontrolled system models [8].

$$\begin{array}{lcl} \rightarrow \{Camera1.on\} & \Rightarrow & Stopper1.Stopped \downarrow \\ & \wedge & Sensor\_Optical1.On \downarrow \\ & \wedge & Area1.P = UNKNOWN \end{array}$$

Figure 8: Example of a requirement model in CIF 3 [8].

event *on* in automaton *Camera1*) when there actually is a non-moving (*Stopper1.Stopped* ↓ denotes that automaton *Stopper1* is in a location with name *Stopped*) workpiece in front of it (*Sensor_Optical1.On* ↓) that has not been scanned before ($Area1.P = UNKNOWN$, where $Area1.P$ refers to the value of the variable P in automaton *Area1*).

Based on the models for the uncontrolled system and the requirements, we tried to apply monolithic controller synthesis techniques to automatically derive a supervisory controller. Due to the large state-space however, this did not succeed. The student is confronted with the issues of state-space-explosion in state-based modelling of concurrent systems and is challenged to find solutions to overcome this challenge. Modular supervisory controllers have been developed by hand in this project. Complete models are available in [8]. In order to validate the correctness of these manually developed supervisory controllers, hybrid (combined discrete and continuous) models are added to represent timing aspects and physical position information of the products. For this hybrid model, a visualization (Figure 9) was added

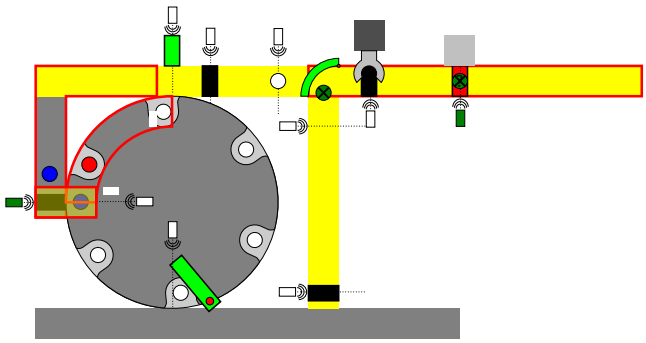


Figure 9: Visualization of the product flow in CIF 3.

to visually validate the controlled system, i.e., the system together with the developed supervisory controllers. The CIF 3 tool provides means to simulate and visualize such hybrid models.

Current research in a PhD project uses xCPS as a case study to develop modular synthesis techniques to overcome or reduce the algorithmic scalability challenges.

3.3 Combinatorial optimization

Supervisory control ensures maximum flexibility of actions while ensuring safe operating boundaries. However, it does not provide any scheduling mechanism that optimizes the sequence of actions to achieve certain performance goals. For example, the goal of xCPS may be to assemble components *as fast as possible*, or to play tic-tac-toe with *the lowest number of moves*. Developing algorithms that generate optimal or close-to-optimal schedules can be challenging, due to the computational complexity associated with such scheduling or optimization problems. Even more so if the optimization needs to be done online, in real-time.

As it is nearly impossible to optimize all aspects of a machine for all use cases, system designers need to make design-time decisions to optimize for certain typical use-cases and operating conditions. Moreover, multiple different objectives play a role and they are often conflicting, for instance productivity, quality and cost. Most designs have some degree of freedom in the high-level configuration of the system. Designers can in that case exploit the reconfigurability towards use cases for different customers.

Besides design-time decisions and system-level configuration, CPS often also require online decision making or scheduling, for instance because the jobs that need to be performed are not known a priori. The quality of such scheduling has an impact on system productivity or processing time. Due to the discrete and often non-linear nature of the system, such scheduling problems are usually of very high computational complexity and solved to near-optimality by online combinatorial optimization algorithms. The computational complexity of the optimization problem is typically highly correlated with the way models are constructed. If complexity allows, it is often insightful to look at offline computed optimal solutions for the resulting models, as this typically gives deeper insight into the behaviour of underlying systems and allows to improve online optimization.

3.3.1 Example combinatorial optimization model

We demonstrate the non-linear behaviour of the design parameters of the xCPS platform over its performance in the example product flow described in Section 3.1. In this example, we show that changing the speed of a belt influences the performance of the system. We assume that the arms take constant time to place a piece or take a piece from a belt. If we now assume that the belts and indexing table are always moving at constant velocity then we can model how the pieces move along the belts or indexing table. The pieces moving over the indexing table have a slightly longer travelling time than those on the belts, as the indexing table's speed is lower. For simplicity, we assume also that the indexing table is a disc that can instantaneously load items at any point in time. The only constraint we impose, is that two pieces need to be at the exact right moment at the pick & place unit to be combined.

```
% Parameters
int: load_time_bottom = 10;
int: load_time_top = 10;

% Decision variables
array [1..n] of var int: load_bottom;
array [1..n] of var int: load_top;

% Constrained to positive time
constraint forall(i in 1..n) (load_bottom[i] >= 0);
constraint forall(i in 1..n) (load_top[i] >= 0);

% Constraints for loading times
constraint forall(i in 1..n) (load_duration_bottom[i] == load_time_bottom);
constraint forall(i in 1..n) (load_duration_top[i] == load_time_top);

% Allow no overlap between the loading of the operations
constraint disjunctive(load_bottom ++ load_top, load_duration_bottom ++ load_duration_top);

% ... other system parameters and constraints ...

% max(unload) is the makespan; set objective to minimize it
solve minimize max(unload);
```

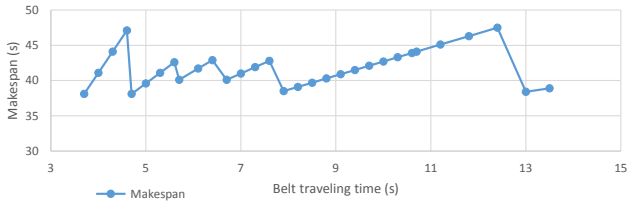
Figure 10: Snippet of the formulation of the system components and constraints in the MiniZinc language

We are interested in finding the loading times of the bottom and top pieces that minimize the makespan of merging and unloading the pieces. To achieve this we have created a MiniZinc [18] model (see Figure 10) to capture the behaviour with a constraint programming formulation. Solving this constraint program yields the optimal loading times. From these loading times we deduce which loading pattern is optimal, considering the travelling times of the pieces along different paths. As an experiment, we have varied the travelling time of one of the segments, i.e., varied the belt speed, to see what impact it has on the total makespan. The results are shown in Figure 11a, which shows on the horizontal axis the assumed travelling time of the belt and on the vertical axis the corresponding optimal makespan. We observe that a *decrease* in the speed of the belt can lead to an *increase* in the performance of the system (i.e. lower makespan), which seems counter-intuitive since slowing down a system component often increases the performance of the system.

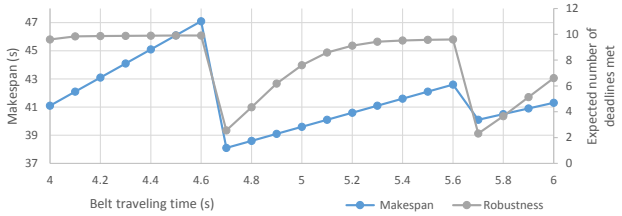
Further investigation reveals how the behaviour can be explained. To show how, we have rendered activity traces for some values of the parameters in the experiment. The two activity traces for different belt speeds are shown in Figure 12 depicting the loading activities of the arm and the time instants that the pieces arrive at a specific location. As a bottom piece is loaded into the system, the loading moment for the corresponding top piece becomes fixed to some future time instant. When the time window permits, we can load an extra bottom piece before we have to load a top piece (see the bottom trace in Figure 12), and we can improve the resulting performance by decreasing the idle time of the gantry arm. However, when there is not enough time between two subsequent loads, no block can be loaded, and the time is wasted idling.

The MiniZinc model of xCPS reveals non-monotonic behaviour of system component speed versus performance. In this particular case, it is important to tune the relative travelling times such that xCPS is effectively utilized. By this activity we have also explored the constraint programming language and solvers, and gained better understanding of the system. Given this model, a system architect can assess the impact of design decisions.

The assumptions used in the MiniZinc model ensure that the domain can be relatively easily mapped to the constraint programming model. Taking into account effects such as



(a) Performance experiment with MiniZinc constraint programming model. Decreasing the speed of the belt may increase the performance of the system.



(b) Robustness experiment: Observing the impact of decreasing belt speeds on schedule robustness. Lower belt speeds and/or poor makespans can result in better robustness. Makespan is on the left axis, while the robustness metric is on the right axis.

Figure 11: Example performance and robustness experiments with combinatorial models.

stopping a belt with multiple pieces on it is harder to model and would lead to a model with higher computational complexity. The discrete slots of the real indexing table also remain to be modelled, but the continuous approximation will yield higher productivity, providing an upper bound on the real performance of the system. Furthermore, the MiniZinc model assumes perfect system components. However, in reality the system operations have variations, the effects of which are investigated in the following section.

3.3.2 *Robust combinatorial optimization*

The signals that control the various sensors and actuators in the xCPS platform are produced by control tasks that are mapped onto general purpose platforms. These platforms suffer from low predictability resulting in variations in task execution times. This uncertain timing behaviour is also apparent in the xCPS platform: not only in the data processing but also in the pick & place actuator that may need a varying number of attempts to pick the pieces using suction in vacuum. There is a need for *robust combinatorial optimization* techniques that generate sequences in which such variations are less likely to lead to significant drops in performance or productivity.

We consider a slightly different variant of the product flow used in Section 3.3.1 so that we can assess the relative robustness of these schedules. We earlier assumed that the tray had constant loading time. In reality there is a variation in the time required to load a piece onto the tray resulting from multiple attempts, or slipping conveyor belts. We model these variations by introducing distributions based on measurements of the load execution times. To study the impact of these variations, we assign deadlines to the merge operations that are 5% higher than their completion times observed in the experiment of Section 3.3.1 and observe the likelihood that these deadlines will be missed due to variations. We analyse robustness of the schedules obtained for

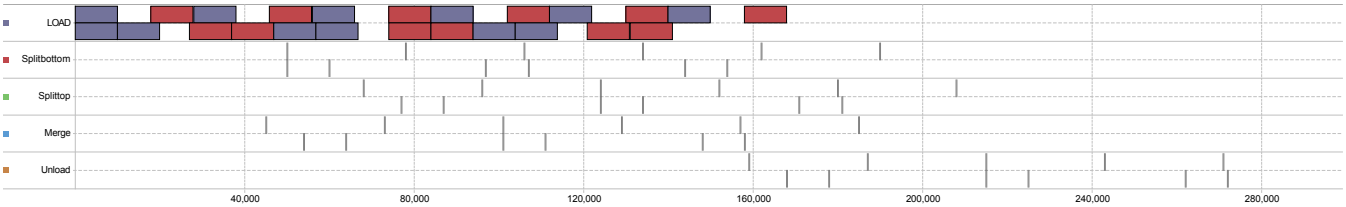


Figure 12: Two activity traces depicting events and loading activities for different belt speeds. Red indicates a top, purple indicates a bottom piece. The bottom trace shows a scenario with lower belt speed than the top trace, allowing two bottoms to be loaded consecutively with reduced idle time compared to the top trace. Time denoted on the x-axis in tenths of seconds.

the range of belt speeds shown on the x-axis of Figure 11b using the technique presented in [2]. Robustness of a schedule is quantified in terms of the expected number of tasks that meet their deadlines in the schedule. This corresponds to the expected number of pick & place operations that do not exceed beyond 5% of their completion times under variations. The higher the value is, the more robust the schedule is. The robustness values are plotted in Figure 11b. The blue graph with y-axis on the left shows again the makespan, while the grey graph with y-axis on the right shows the corresponding robustness. We observe that schedules with larger makespans are often more robust owing to the presence of gaps in the schedules to absorb the variations. Considering such combinations of optimization criteria such as makespan and robustness to make design decisions is an even harder challenge requiring efficient trade-off analysis.

3.3.3 Application in industry and education

The two experiments of Sections 3.3 and 3.3.2 are performed on a relatively simple part of the xCPS platform. Such experiments in a real system can become quite complex to interpret and therefore these experiments give a hint of the complexity of the optimization problems in real CPS. Even in these simple examples we reveal trade-offs that can be influenced by controller design, physical layout, and scheduling decisions. It is essential for the performance of high-tech systems that such a multi-disciplinary view is adopted at an early stage in the design process.

The xCPS platform can be used to simulate real industrial machines such as chip fabrication machines and large-scale production printers. This kind of industrial machines perform different actions taking different amounts of time: changing lithography masks or adjusting the print head height for thicker/thinner paper respectively. We can simulate such behaviour by constraining the execution of certain physical tasks on the xCPS platform. For example, the red item takes ten seconds to process, while the black and silver components can be processed in five seconds. Some machines also take some time to re-adjust between processing different (kinds of) items; e.g. a reconfiguration penalty of five seconds may be imposed when processing a black item directly after a silver item. The order in which these actions have been scheduled can heavily influence the performance. This behaviour is known as *sequence-dependent setup* and is a crucial aspect in scheduling problems such as production printing [31] and wafer scanners [1].

The xCPS platform is used to teach students different aspects of performance modelling and combinatorial optimization. The students can learn which aspects of the optimization problem are essential to achieve a high quality model. For example, choosing which decision variables to take into account in what way, which solution-space search strategy works better and determining the selection of derived constraints to reduce the search space. These aspects are usually key in getting good solutions to the optimization problem in a limited time budget.

xCPS facilitates exploration of the strong and the weak points of different modelling and optimization approaches. For example, we can use Synchronous Data Flow (SDF) [13], genetic algorithms [9] [6], (meta-)heuristics [6] and constraint solving based approaches [18] to find schedules for the xCPS platform. Comparing these approaches on an industrial high-tech system is very challenging due to the complexity and/or even impossible due to intellectual property restrictions or cost. The xCPS platform shows key ingredients that are of interest for an industrial system or can be easily extended to incorporate such components. Optimizing machines with e.g. setup times, buffering, pipelining, and/or timing variation, in a reduced complexity setting allows students to explore the combinatorial challenges of modern high-tech systems.

3.4 Timing Analysis

Manufacturing systems have key-performance indicators based on throughput and latency, e.g. the number of products produced per hour while satisfying certain deadlines in the system. Latency metrics are defined as the time distance between two specified events in the system, for instance between the start of processing and the output of a product. These metrics must be considered in the design-space exploration, scheduling decisions, or for validation purposes. Therefore, timing analysis of CPS is an important subject to study or research, which can be facilitated with the xCPS platform.

For xCPS, we look at performance analysis at the system level and at the resource level. The particular challenge is to capture the timing behaviour of tasks in the product flow taking into account the tasks themselves, the resources they use, as well as their mutual dependencies. For example, the product flow relies on synchronizing events between different actions of actuators or sensors. This can be modelled as tasks with dependencies. Such systems also have resource dependencies when resources are shared between multiple tasks. Besides task and resources dependencies, these systems depend on the pipelined processing of multiple products to increase performance. This causes additional dependencies between subsequent products, for example while one product is being assembled, a second product can already start to be processed.

3.4.1 Example of SADF modeling of xCPS

We are exploring the use of data flow models-of-computation, such as Synchronous Data Flow graphs [13] or extensions of this model such as Scenario-Aware Data Flow (SADF) [27], since they can naturally capture (cyclic) task dependencies, resource dependencies and pipelined behaviour. Task durations (execution times) and synchronizations are also natural ingredients of data flow models. Moreover, data flow models have good analyzability properties due to their deterministic and time-monotone behaviour. For the analysis we employ our research data flow analysis tool SDF^3 [24] to determine throughput and latency.

In a data flow model an actor represents a task (usually a small computation). Edges between actors represent task-dependencies where data is transmitted through tokens, modelled by black dots on edges. Each input and output port of an actor has a set of rates that defines the number of tokens consumed or produced by that actor. An actor *fires* when all tokens are available at the input edges of the actor. When an actor fires, it consumes a number of tokens matching the rate of each input edge. When the firing is finished the actor produces a number of tokens matching the rates of each output edge. This makes this abstraction suitable to model concurrency in applications with cyclic, direct data dependencies and pipelining dependencies. For the models in this paper we assume all rates of the ports to be one, i.e., each port consumes and produces a single token. We consider the tokens to model one piece. This way we can view the token flow in the data flow graph as a model of the flow of pieces in the xCPS system. However, modelling resource sharing is not trivial. For this reason, we use a more expressive data flow model, SADF. SADF permits the use of different scenarios to express more dynamic aspects of applications, such as different execution times, port rates, but particularly also resource arbitration decisions.

As an example of what can be studied using SADF models of the xCPS system, consider the example described in Section 3.1, where we assume that the objective is to assemble a piece, composed of a top and bottom piece, as fast as possible. For this purpose, the system has a decision point on *conveyor belt 2*, where using a pusher a piece can either be passed onto *conveyor belt 3* (bottom) or to the *index table* (top). We demonstrate how to model this example using SADF and considering fixed execution times for tasks, maximum capacity of resources and a fixed sequence of of inputs for the pieces (input a bottom always before a top).

We model these different flows for bottom and top pieces as two different scenarios. In Figure 13a we capture the flow of a bottom piece. The *src* actor models the input of a piece, the *c2a* actor is the task of transporting the piece on the conveyor belt 2 (up to the stopper) and the *index* actor is the task of transporting the piece on the index table 2. In Figure 13b the flow of the top piece is captured. The flow is the same for actors *src* and *c2a*, but reaching the decision point in this case, the piece keeps flowing through the rest of conveyor belt 2 onto conveyor belt 3, instead of being passed over to the index table 2. This is captured by actors *c2b* and *c3a*, modeling the conveyor belt 2 after the stopper and the portion of conveyor belt 3 up to the pick-and-place unit. Finally, actor *p-p* models the action of the pick-and-place unit, where the two pieces are assembled.

Since we require that both the *src* and the *p-p* are synchronization points, we can only insert one piece at a time and the assemble operation requires the two blocks to be ready. We model such inter-scenario dependencies using *persistent labelled tokens*, tokens that can be shared between different scenarios in an SADF model. Therefore, we have two distinct types of tokens in our model. Unlabelled tokens model the piece flow, labelled tokens model synchronization or resource dependencies. In an SADF model, if a token is consumed by an actor in one scenario, this token becomes unavailable in all scenarios of the model. Therefore, if we add a persistent token on the self-cycle of actor *src* in both scenarios, this forces that there is no concurrency between *src* actors of different scenarios. The same holds for the shared resource of conveyor belt 2, actor *c2a* is used in both scenarios, modelled by persistent token b. The last case, the synchronization of two pieces at the pick-and-place, is in this case modelled by a persistent token c. This token is produced after each execution of the actor *index*, symbolizing the piece reached the pick-and-place unit in Scenario 1, whilst in Scenario 2 this token symbolizes a requirement, that in order for actor *p-p* to fire, it requires token c to have been produced. This, however, forces that for the correctness of the model, the scenario execution order needs to be fixed. In this case, Scenario 1 needs to always precede an execution of Scenario 2. In SADF this can be expressed in the finite-state machine that defines the possible orderings of Scenarios within the model, Figure 13c. We could relax this assumption by using a different modelling approach, such as the one in [5]. We model resource sharing and capacity in a similar fashion. Persistent labelled tokens labelled *c2a1,c2a2* are used to model the limited capacity of conveyor belt 2, such that a maximum of two pieces can be simultaneously on the belt.

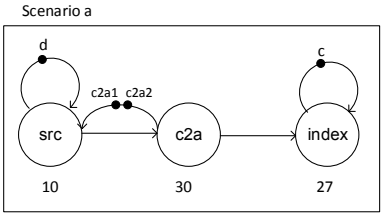
3.4.2 Timing Analysis using SADF

With the SADF model we can analyze the maximal achievable throughput for the fixed input bottom-top, using the SDF^3 data flow analysis tool. For this particular example model, we get a value of 1/60 pieces per time unit. Moreover, the tool can perform a simulation of a selected sequence of scenarios to obtain a Gantt chart that visualizes the behaviour and the resulting execution trace can be studied to obtain parameters such as latency, throughput and to discover potential bottlenecks. Figure 14 depicts the Gantt chart of the system for a scenario sequence of 4 pieces $\{a, b, a, b, a, b, a, b\}$. The execution of the actors in the same scenario and also same iteration are shown with the same colour. The latency of a complete assembly of a piece, is the time difference between the start time of the first actor *src* firing (i.e., entrance time of the first bottom object) and the end time of the last actor *p-p* firing (i.e., output time of the last assembled object). As shown in Figure 14, for example, the latency of assembling the second object equals 116 time units. The throughput of this system is equal to the average firing rate of actor *p-p* in Scenario b, which is depicted in the last row of the Gantt chart. Furthermore, it is also possible to obtain the bottleneck of the system. In this example, it is visible that actor *index* creates a bottleneck in the system, i.e. by decreasing the execution time of actor *c2a* from 27 to 15 the throughput of the system increase from 1/60 to 1/45.

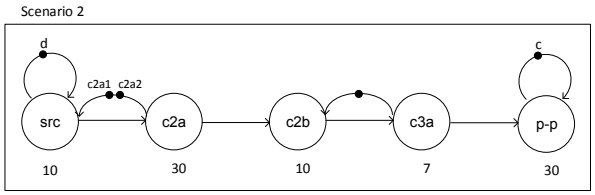
With this SADF model an optimal or safe order can be derived using combinatorial optimization or controller syn-

thesis techniques as discussed in other sections. Data flow analysis results can then be used to find and resolve bottlenecks within xCPS. This project is being pursued by a PhD student. Another PhD student considers the use of data flow models in design-space exploration. A data flow model is created that abstractly represents the whole system. It can then be used to explore different platform configurations (e.g., number of resources, types of resources) and find the optimal platform(s) based on one or more optimization objectives (timing performance indicators or resource cost). For instance one may explore the cost/performance trade-offs from using faster or slower conveyor belts or processing units. From these trade-offs one could for instance determine the lowest number of processing units to meet a certain throughput requirement. In the xCPS platform, the use case is the design of input/output of products using the pneumatic arms (Gantry arms 1 and 2 in Figure 3). Currently, the objective is to model the input/output flow using SADF [27] to estimate product flow performance and explore different platform designs considering speed of the arms (using different profile settings) or the use of both or only one arm.

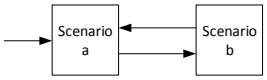
The above analysis techniques use worst-case task execution timings, assumed to be known beforehand. Predicting the timing behaviour of the xCPS system heavily depends on the accuracy of the timing models. Dealing with unpredictable platforms with large, stochastic variations, raises the need for timing models to also take timing variations into account giving rise to the PhD study of *robust timing analysis*. Industrial CPS often have timing information limited only to measurements. We need efficient techniques to obtain these measurements and combine them into reasonable estimates of execution time distributions. Another challenge is that most performance measures are derived from the completion times of operations. Obtaining these completion time distributions needs complex stochastic analysis involving analytically hard *max* operations on distributions [2]. We have studied the statistical analysis [1] of this behaviour. With that information robustness analysis can be employed to compute metrics which quantify the robustness of the system towards achieving desired performance. An example of a stochastic metric is the expected value of the number of pieces assembled by xCPS per hour, as illustrated in Section 3.3.2.



(a) Bottom piece scenario



(b) Top piece scenario



(c) Finite-state machine

Figure 13: SADF Model

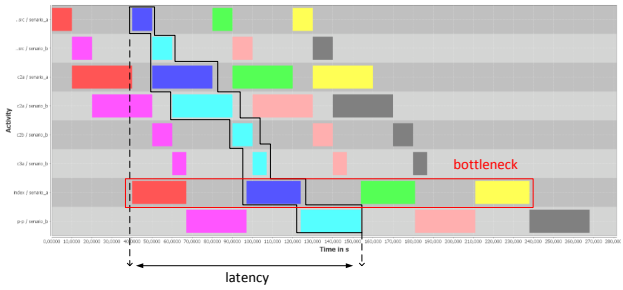


Figure 14: Trace of SADF model

3.5 Automatic Verification

In this section we turn to a different type of analysis, namely the use of automatic verification techniques. We also move to a different level of abstraction compared to the previous sections. Previously we considered course-grained actions such as transport on a belt as indivisible tasks with an *assumed* fixed (possibly worst case) execution time. Under more detailed inspection, the execution of such an action or task is the result of the dynamics of the underlying subsystem, i.e., the actuator executing the task, possibly in feedback control, and affected by noise, errors and physical disturbances. Therefore, the higher level abstractions are valid only if the system sufficiently accurately satisfies the assumed properties. Checking this conformance can be achieved with verification techniques.

Specifically, in this section we zoom into the details of one of such actions. We explore the use of abstraction techniques for infinite-state and hybrid automata to *verify* that the properties assumed by the higher level models are respected by their physical realization. To this end we employ automata-based abstraction and verification techniques to verify the validity of these properties.

To study the behaviour under all possible control inputs, we need a suitable model of the system. In control theory, it is common to derive a state space model of the dynamical systems, in which the states represent the relevant physical quantities of the system by real valued numbers. This model however is not directly amenable to automata-based verification techniques such as model-checking. The continuous variables give rise to a system with an (uncountably) infinite number of states and verification requires all reachable states to be enumerated in finite time. This hurdle can be overcome by making finite-state abstractions of the infinite state model [25].

We present a simplified example that leads to a model with

only a few states, so we can visualize it and illustrate the concepts involved. We consider the task of moving a piece to a desired place (for instance at a turner) by a conveyor belt. We consider the position x of the piece on the conveyor belt to be a model of the (continuous) state of the system. We assume the belt has only one possible control input (actuator action): turning on the conveyor belt for a fixed period of time. This action makes the piece move, and hence changes the state x to a new state x'. Due to small (bounded) physical variations there is some disturbance, variation, in x'. The verification question is whether it is possible to control the belt to leave the piece sufficiently accurately at the desired position.

The finite abstraction applied to achieve a finite state-space for analysis consists in a partitioning of the continuous state-space (piece positions on the belt) that is sufficiently precise to allow us to distinguish being positioned at the desired place from being positioned at a wrong place. The unpredictable variations in the state due to disturbances leads to non-deterministic changes of the system state in the state-spaces. Our analysis goal is to prove that we can guarantee the placement of pieces at the desired place, despite the disturbances. To express this quality of the system we define the following formal properties:

- Property 1: There exists a sequence of control inputs such that the piece stops at the desired place somewhere during the sequence.
- Property 2: There exists a sequence of control inputs such that the piece ends up at the desired place at the end of the sequence.

Property 1 states that irrespective of the error introduced by variation in the response to control input, the piece ends up at the desired place somewhere during the sequence, at the end of one of the individual inputs. If this property is satisfied then we can make sure that for all pieces starting from the initial state, the piece will end up at the desired place and can be pushed/rotated. We would need a sensor to know exactly at what step in the sequence the place is reached. If no such sensor is available however, we require the stronger Property 2. Property 2 states that we can guarantee that *at the end* of the sequence the piece is in the right place, irrespective of the random variation that it experienced. In that case the sensor is not required to confirm when the piece is in the right location.

Figure 15 shows finite state models of the belt controlled by three different controllers having different activation times for the belt, after abstraction. To construct these models it is assumed that the conveyor has an unknown speed error, but within a known bound. Abstract states in which the piece is positioned in the desired spot are shown in green. The states in yellow represent positions on the belt before the target and the red ones are states in which the piece may have passed the target position, in which case the target can no longer be reached. Figure 15a shows that there exists a behaviour in which the piece never stops in the desired positions (passes a green state). The activations are too long and the piece may go at once from a position before the target to a position after the target (according to the abstracted model). In Figure 15b a smaller activation period

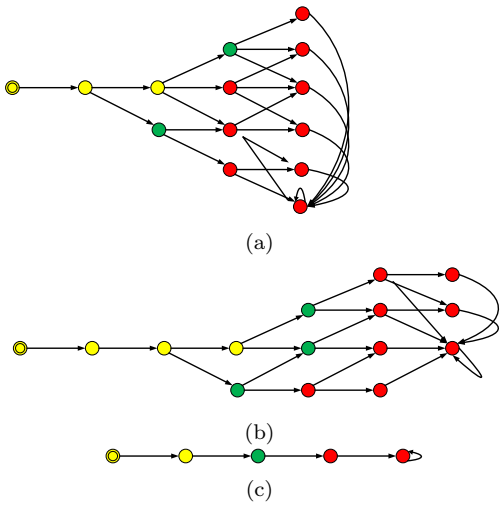


Figure 15: Finite state models of the belt for three different controllers

is used. We can see that the piece will eventually stop in a green state, either after three or after four activations, i.e., Property 1 is satisfied. However, neither a sequence of three activations nor a sequence of four activations are guaranteed to bring the piece to the right place at the end of the sequence, so Property 2 is not satisfied in this model. Finally, Figure 15c shows the abstract state-space according to a different activation period, which is suitable for a feed forward controller without sensor; after two activations the piece is guaranteed to be at the desired position (Property 2 is satisfied). As we have now verified the behaviour of this controller we may choose to implement this controller and we know the fixed execution time of the task.

In this small example we were able to visually check for the desired properties of the system. However, in general, the finite-state models can have many more states and therefore need model-checker tool support to check its properties automatically. To this end, the model checkers require the properties to be expressed formally in some *logic*. Since our properties qualify in terms of sequences of states, we use *modal (temporal) logic* [19] to express them. Modal logic can express statements like whether eventually something good must happen (liveness properties) or that something bad will never happen (safety properties). For example, Property 1 is a liveness property.

Expressing desired properties in a formalism that allows for tool-based automatic verification is not always straightforward. Translating physical properties into precise statement in formal logic is an instructive challenge with educational value. Moreover, different verification tools support different classes of logical properties to be checked and different types of properties have different complexity to be checked. We have used the LTSA tool [15] to automatically check the validity of our properties, because it supports our required property types.

3.6 Image-based sensing control

Image-based controllers use sensor data that is extracted from camera images by an image processing algorithm. As the computational power of embedded devices increases,

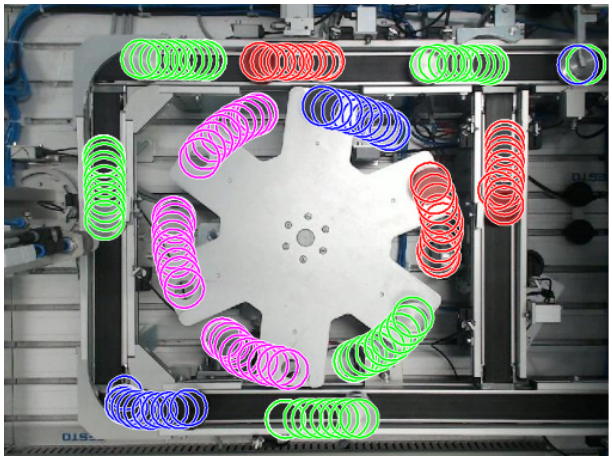


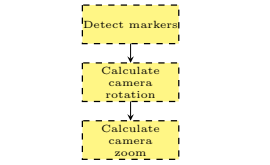
Figure 16: The *Matlab* version of the image processing algorithm working on a pre-recorded video stream, currently tracking 11 pieces. The tracks shown are from the first 12 frames of the video. The circles of the colours correspond to the object types. Purple denotes an empty object location on the indexing table.

these controllers are becoming more common in CPS such as robotics and Advanced Driver Assistance Systems (ADAS). In ADAS, an image sensor is used to detect, track and classify objects. This information is used to either help the driver control the vehicle or to autonomously control it [12]. In robotics, image based sensing is used to track a reference (objects of interest) and guide a robot towards it [7].

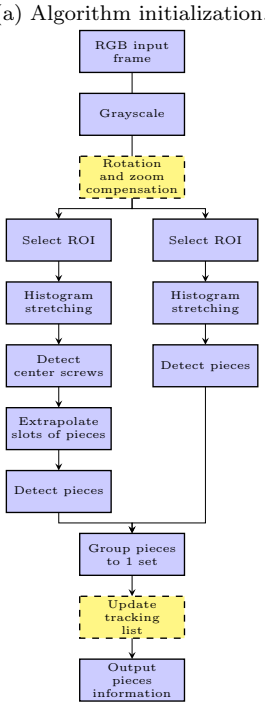
Image based control involves multiple challenges. Firstly, the image processing algorithm should have both a reliable quality and a bounded execution time. Secondly, trade-offs between quality of sensing, resource usage and quality of control play an important role. Finally, the controller should be able to cope with the latency induced by the image processing algorithm providing performance guarantees. We are studying and exploring these challenges in the xCPS platform by trying to control the assembly/disassembly process with a camera through multiple student projects.

3.6.1 Developing image processing algorithms

In a bachelor level project an image processing algorithm was implemented in Matlab. Such algorithm obtains the position of all the objects in one section of the *product flow*, for example, the conveyor belts and the indexing tables [30] (see Figure 16). The algorithm is summarized in Figure 17b. The image is greyscaled and two Regions of Interest (ROI) are extracted: one containing the conveyor belts and the other the indexing tables. The histograms in both ROIs are stretched to compensate for the differences in illumination. In the indexing table, a preprocessing procedure detects the screws in the centre and extrapolates the position of the pieces slots. The pieces on both parts of the image are located using a Hough Transform for circles. The execution time of the algorithm is one second. The quality of the algorithm was evaluated using $F1$ score [22]. This score measures the quality of an algorithm considering false positives, false negatives, true positives, and true negatives, in a range varying from zero (no detections) to one (perfect detections). The average $F1$ score for the Matlab algorithm is 0.9428. Despite of this high score, the algorithm leads



(a) Algorithm initialization.



(b) Operations in each frame

Figure 17: Comparison of Image processing algorithms for pieces detection. Yellow dashed blocks are only available in the C implementation. Blue blocks are available in both implementations.

to a slow assembly/disassembly process because of its relatively long execution time. A successor internship project ported the image processing algorithm to the programming language C and extended its functionality. The comparison of both implementations is shown in Figure 17b. The C implementation added an initialization procedure (see Figure 17a) which corrects for differences in zoom and rotation of the camera, based on the detection of markers on the machine. A tracking list of the detected pieces is also added. This functionality tags all the detected pieces, so that false positives and false negatives are identified and corrected. The execution time of such software is $10ms$. The average $F1$ score is 0.8835.

$F1$ scores for both implementations are compared on a pre-recorded video in Figure 18. The *Matlab* implementation outperforms the C implementation, because the libraries containing the Hough transform are different in both Environments. Therefore the C implementation shows more false positives than the *Matlab* implementation. However, it is important to highlight that the C implementation executes 100 times faster than the *Matlab* implementation. A successor internship project considers implementing the current image processing algorithm in an embedded platform.

3.6.2 *Exploring image-based control trade-offs*

Trade-offs between quality of sensing, resource usage and quality of control are also being studied. A master thesis project is analysing these trade-offs arising from alternative image processing algorithms: one slower and more accurate and one faster but less accurate. The goal of this project is to improve the quality of control by developing an adaptive controller that switches between the two algorithms (Figure 19), instead of using a single one.

An initial experiment was carried out to improve the quality of control of one of the conveyor belts in the *product flow* example. The experiment considered a fixed control strategy for both image processing algorithms and a limited number of pieces. The decision diagram shown in Figure 20 was used to switch between the sensing algorithms. The diagram considers the use of the fast algorithm when the error is larger than a threshold and the slow algorithm otherwise. The threshold is found by running a set of simulations with different thresholds values. The chosen value minimizes the integral of the absolute error of the control loop. Figure 21 shows the performance of the controller following a step reference with three changes and multiple sensing algorithms. The fast algorithm does not reach the reference due to the sensing errors. The slow algorithm shows the largest overshoot in the graph because the sensing latency forces the controller to actuate based on old information. The accuracy of the slow algorithm allows the controller to reach and stay at the reference. Figure 21 also shows that the adaptive sensing strategy outperforms the use of a single sensing algorithm: the overshoot is virtually equal to fast algorithm and steady state response equal to the slow algorithm. This improvement is explained because when the system is close to the reference (inside the threshold), the controller can drive the plant to the set-point within a few outputs; to do this accurate sensing information is needed. On the other hand, when the system is far from the reference (outside the threshold), the controller needs a large number of controller outputs to drive the system in to reference; for this accuracy is less important. Extensions of this work include the design of a decision diagram with more than one object and takes into account the resources usage, the design of controllers that compensate for the sensing latency of each algorithm and the application of the analysis to other sections of the machine.

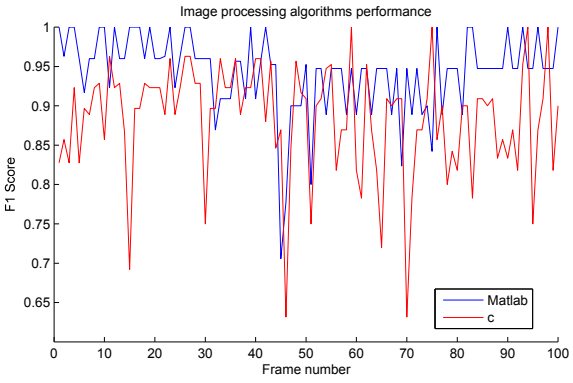


Figure 18: F1 score comparison between the two implementations of the image processing algorithm

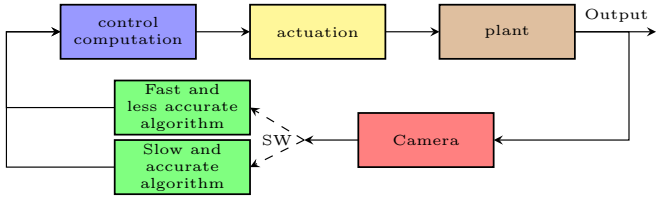


Figure 19: Image based control with two possible image processing algorithms

3.6.3 *Coping with sensing latency*

The development of control systems capable of dealing with the sensing latency is being investigated in a PhD research project. A classical hardware implementation of a control system with image sensing (e.g. Figure 22a) produces a sample period larger than the sensing latency and therefore a low control performance. An alternative to cope with this latency is to pipeline the sensing algorithm (e.g. Figure 22b). Pipelining the sensing algorithm allows for a reduction in the sample period and an improvement of the control performance.

In xCPS, a long sensing latency is present because a camera and an image processing algorithm are used as sensor in the control loop that regulates the speed of the assembly pieces. A simulation comparing the performance of controllers with a classical configuration and a two-core pipelined sensing configuration is shown in Figure 23. The pipelined sensing generates twice the sensing information samples (red round markers) compared to the classical sensing (blue diamond markers). The classical controller reaches the reference in 0.21 seconds whereas the pipelined controller reaches it 17% faster.

A pipelined implementation leads to a trade-off between hardware usage and quality of control, because it potentially requires a large amount of resources in order to achieve a significant reduction in the sample period. That is why, the research is concerned not only with coping with the sensing latency, but also of cross-layer co-design between hardware resources and the quality of control strategies for image-based sensing, a typical concern in CPS.

4. DISCUSSION

Different aspects of design and analysis have been highlighted in the previous sections. Examples of various activities have been given and it has been shown how many of these aspects are interdependent and interrelated. Espe-

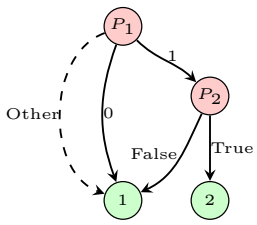


Figure 20: Decision diagram. P_1 verifies the number of objects in the picture. P_2 verifies if the error is within the threshold. 1 and 2 correspond to the fast and slow image processing algorithms respectively.

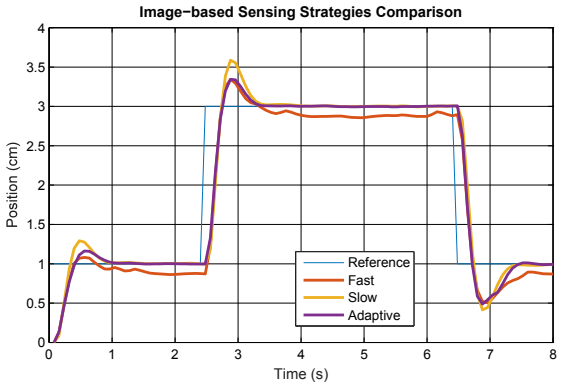


Figure 21: Sensing algorithms comparison

cially in design of CPS, learning how to effectively master these interrelationships to come to an overall optimal system level design is of crucial value.

At the highest abstraction level, the supervisory control constructs the state space of all safe and deadlock free configurations (built from abstract states and coarse grain actions). Combinatorial optimization techniques are used to explore the state space of safe configurations to find the optimal behaviours and optimal (run-time) scheduling strategies. Given a schedule, timing analysis techniques provide the time-based performance metrics such as throughput or

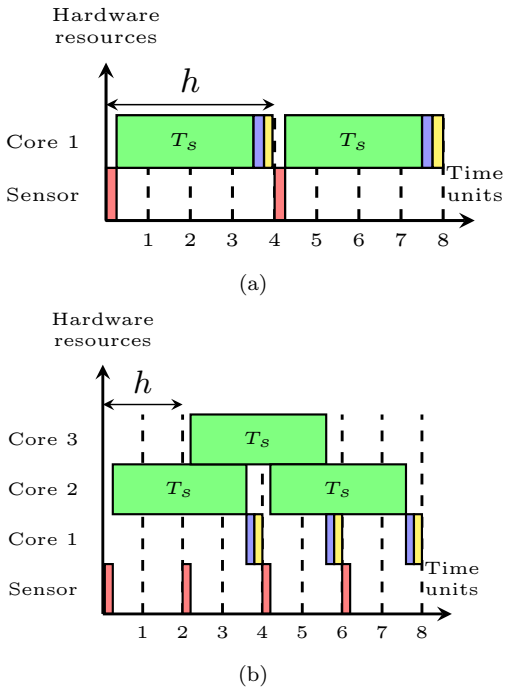


Figure 22: Hardware resources in image-based control. a) Example of a classical implementation. The sensing (green blocks), control computation (blue blocks) and actuating tasks (yellow blocks) are executed in a single core. The sample period in this example is $h = 4$ time units. b) Example of pipelined implementation with 2 sensing cores. The sample period is reduced to $h = 2$ time units.

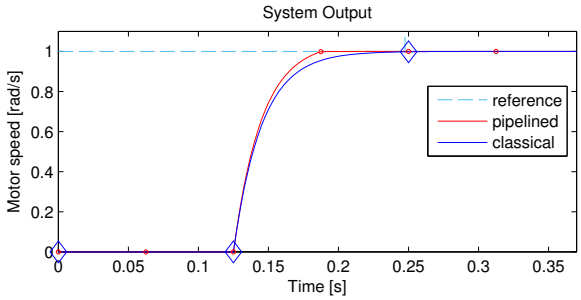


Figure 23: Comparison between controllers with classical and two-core pipelined implementations.

latency. At the lowest abstraction level, continuous feedback controllers implement the abstract actions and verification and synthesis techniques guarantee that the higher level abstract models of the actions are sound, i.e., adhere to the physical world behaviour.

xCPS as a platform facilitates research and learning on all these aspects. Furthermore, the various activities on the platform are often multidisciplinary in nature and encourage collaboration between students with different backgrounds, in Computer Science, Computer Engineering, Electrical Engineering and Mechanical Engineering, with the side-effect of also developing non-technical skills challenged by such cooperations.

5. RELATED WORK

Recently there has been a lot of interest in the development of CPS platforms to be used both for research and teaching. The xCPS platform presents more general concerns than the automatic control platforms where the main goal is to verify and experiment with the functionality of traditional feedback control solutions. Axelsson et al. [3] introduce MOPED, a mobile open platform for experimental design of CPS. The platform consists of a model car chassis, controlled by a set of three control units running the automotive software standard AUTOSAR. It is designed to be highly representative of real automotive systems in terms of software, while simplifying other aspects. The platform is extensible, to allow both students and researchers to add new functionality and interfaces. A robot car platform is developed by González-Nalda et al. [10], where a Raspberry Pi board is used as processing unit and there is interaction with the robot using WiFi.

In xCPS, image-based controllers are used as part of some controllers. Mosterman et al. [17] introduce a manufacturing facility that solves the Towers of Hanoi puzzle, where image processing is also used in the control loop. Compared to xCPS, there is also more focus on feedback and feed-forward control for the actuators. The main aim of the platform in [17] is to be used by students in project-based learning whereas xCPS provides a flexible platform imitating industrial scale manufacturing systems.

CPS education is also provided in the form of a massive open online course (MOOC) [11]. Here, the iRobot Roomba autonomous vacuum cleaner is used as a platform for CPS research. The platform is capable of driving, sensing the surroundings, executing scripts and communicating with an

external controller. Similar to xCPS, a hardware abstraction layer is provided for programming. This layer allows simulation of sensors and actuators within a simulator, and linking executable models to the actual sensor and actuator drivers. Although xCPS employs several similar techniques, the approach focuses on imitating manufacturing systems, which shows a wider range of challenges than focusing on the implementation level. This puts more emphasis on ways to cope with different models and different abstraction levels.

Another line of research focuses on physical and virtual education platforms for automatic control. The e-puck [16] is a low-cost desktop size mobile robot designed for use in an educational context. This physical device is used to teach signal processing, automatic control, distributed intelligence, and path finding [16]. It focuses on automatic control approaches and teaching how to implement these. xCPS focuses more on the interaction between the disciplines; the trade-offs between mechanical design, controller design of the platform, and scheduling computational activities.

6. CONCLUSIONS

This paper presents the xCPS research and education platform, that is designed to be representative of CPS with industrial-size complexity. The platform contains various types of sensors and actuators that allow many different types of challenges to be investigated and CPS related competencies to be explored and developed. From an educational and research point of view, students working with the platform not only become familiar with the different types of subsystems, their design trade-offs and models, methods, analysis and tools, but they also experience how the parts interrelate and the importance of cross-layer design considerations that are typical of cyber-physical systems. More information about the xCPS platform is available at `www.xcps.info`.

Acknowledgments

This research is supported by the Dutch Technology Foundation STW under the Robust CPS program (projects 12693, 12694, and 12697) and the NEST project (10346), by the ARTEMIS joint undertaking under the ALMARVI project (621439), and by ASML, Océ, Technolution, TNO-ESI, and FEI.

References

[1] S. Adyanthaya, M. Geilen, T. Basten, R. Schiffelers, B. Theelen, and J. Voeten. Fast multiprocessor scheduling with fixed task binding of large scale industrial cyber physical systems. In *Digital System Design (DSD), Euromicro Conf. on*, pages 979–988, 2013.

[2] S. Adyanthaya, Z. Zhang, M. Geilen, J. Voeten, T. Basten, and R. Schiffelers. Robustness analysis of multiprocessor schedules. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), Int. Conf. on*, pages 9–17, 2014.

[3] J. Axelsson, A. Kobetski, Z. Ni, S. Zhang, and E. Johansson. Moped: A mobile open platform for experimental design of cyber-physical systems. In *Software Engineering and Advanced Applications (SEAA), 40th EUROMICRO Conf. on*, pages 423–430, 2014.

[4] J. Baeten, J. van de Mortel-Fronczak, and J. Rooda. Integration of supervisory control synthesis in model-based systems engineering. *Proc. of Special Int. Conf. on Complex Systems: Synergy, of Control, Communications and Computing*, pages 167–178, 2011.

[5] J. Bastos, S. Stuijk, J. Voeten, J. Jacobs, R. Schiffelers, and H. Corporaal. Modeling resource sharing using SADF. In *Formal Methods and Models for Codesign (MEMOCODE), 13th IEEE/ACM Int. Conf. on*, 2015.

[6] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, pages 82 – 117, 2013.

[7] L. Feng. *Development and applications of a vision-based unmanned helicopter*. PhD thesis, 2010.

[8] R. Fransen. Modeling a flow shop workstation using CIF. Bachelor thesis, Eindhoven University of Technology, 2015. CST 2015.077.

[9] D. Goldberg. *Genetic Algorithms*. Pearson Education, 2006.

[10] P. Gonzalez-Nalda, I. Calvo, I. Etxeberria-Agiriano, A. Garcia-Ruiz, S. Martinez-Lesta, and D. Caballero-Martin. The challenge of building a cyber physical system as an educational experience. In *Information Systems and Technologies (CISTI), 9th Iberian Conference on*, pages 1–6, 2014.

[11] J. C. Jensen, E. A. Lee, and S. A. Seshia. Virtualizing cyber-physical systems: Bringing CPS to online education. In *Proc. First Workshop on CPS Education (CPS-Ed)*, 2013.

[12] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo. Development of Autonomous Car - Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture. *IEEE Trans. on Industrial Electronics*, 0046:1–1, 2015.

[13] E. Lee, D. G. Messerschmitt, et al. Synchronous data flow. *Proc. of the IEEE*, 75(9):1235–1245, 1987.

[14] Y. Li., J. Voeten, and R. Frijns. A model-driven design approach for an industrial-scale mechatronic system. Master thesis, Eindhoven University of Technology, 2011.

[15] J. Magee and J. Kramer. *State models and Java programs*. Wiley, 1999.

[16] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proc. of the 9th conference on autonomous robot systems and competitions*, pages 59–65, 2009.

[17] P. Mosterman, J. Zander, and Z. Han. The towers of hanoi as a cyber-physical system education case study. *Proc. of the First Workshop on Cyber-Physical Systems Education at CPSWeek*, 2013.

[18] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming* , volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer Berlin Heidelberg, 2007.

[19] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[20] P. J. G. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J.*

Control Optim., 25(1):206–230, 1987.

[21] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, 1989.

[22] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, 2003.

[23] M. Skoldstam, K. Åkesson, and M. Fabian. Modeling of discrete event systems using finite automata with variables. In *Decision and Control, 46th IEEE Conf. on*, pages 3387–3392, 2007.

[24] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference Proceedings*, pages 276–278. IEEE, 2006.

[25] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[26] B. Theelen, O. Florescu, M. Geilen, J. Huang, P. van der Putten, and J. Voeten. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. *Formal Methods and Models for Codesign (MEMOCODE), 5th IEEE/ACM Int. Conf. on*, pages 139–148, 2007.

[27] B. D. Theelen, M. Geilen, T. Basten, J. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Formal Methods and Models for Codesign (MEMOCODE), 4th IEEE/ACM Int. Conf. on*, pages 185–194, 2006.

[28] D. A. van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers. CIF 3: Model-based engineering of supervisory controllers. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 575–580. Springer Berlin Heidelberg, 2014.

[29] B. van der Sanden, M. Reniers, M. Geilen, T. Basten, J. Jacobs, J. Voeten, and R. Schiffelers. Modular model-based supervisory controller design for wafer logistics in lithography machines. In *Proc. of MODELS*, 2015.

[30] S. Vegt. A Fast and Robust Algorithm for the Detection of Circular Pieces in a Cyber Physical System. Bachelor thesis, Eindhoven University of Technology, 2015. available at www.es.ele.tue.nl/esreports/esr-2015-02.pdf.

[31] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. Corporaal. A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer. In *DATE, Proc. of the*, pages 573–578, San Jose, CA, USA, 2015. EDA Consortium.