Control-Quality Driven Task Mapping for Distributed Embedded Control Systems

Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng
Department of Computer and Information Science
Linköping University, Sweden

***Abstract*—Many embedded control systems are implemented on execution platforms with several computation nodes and communication components. Distributed embedded control systems typically comprise multiple control loops that share the available computation and communication resources of the platform. It is well known that such resource sharing leads to complex delay characteristics that degrade the control quality if not properly taken into account at design time. Scheduling in computation nodes and communication infrastructure, as well as execution periods of the controllers impact the delay characteristics and, consequently, the control quality. In addition, mapping of tasks on computation nodes affect both scheduling of tasks and messages, and the assignment of periods of the control applications. Therefore, control synthesis must be considered during mapping, scheduling, and period assignment in order to achieve high control quality. This paper presents a control-quality optimization approach for integrated mapping, scheduling, period selection, and control synthesis for distributed embedded control systems.**

I. Introduction and Related Work

The development of embedded control systems comprises two main activities. First, control algorithms are developed and their rates of execution are decided. Second, the control applications are implemented as periodic tasks on a given platform, which in most modern systems comprises several computation nodes and appropriate communication interfaces. These periodic tasks perform sampling and measurements, as well as computation and actuation of the control signal; the tasks may also implement other required operations (e.g., any application-specific data processing on the inputs and outputs). Automotive and avionics systems are two typical representatives of application domains in which multiple controllers execute on a shared, distributed computation platform. Traditionally, the two mentioned activities have been treated separately, and in practice even by different engineering teams.

The control algorithm—also referred to as the control law—is developed based on a model of the physical process or plant to be controlled [1]. The sampling and actuation period is decided based on the dynamics of the plant and possibly also by the available computation and communication bandwidth. The control law is typically synthesized and customized for the chosen sampling period. Although a constant sampling–actuation delay can be taken into account during control-law synthesis, delays and their runtime variation may degrade quality of control, and in the worst-case jeopardize stability. The two implementation-related timing properties that affect the control quality are the sampling period and delay characteristics (e.g., average and variance of the delay) [2], [3].

Once control algorithms and their execution periods are decided, all control tasks are implemented on a distributed platform. First, each task is mapped to a computation node, and second, schedule parameters of tasks and messages are decided for the scheduling policy and communication protocol of the platform. If time-triggered execution and communication is supported by the platform [4], then the scheduling step is to synthesize schedule tables with start times for task executions and message transmissions. For platforms with fixed-priority scheduling [5] and a CAN bus [6], this step involves an assignment of task and message priorities.

In addition to the computation delay of a control application itself, resource sharing and communication contribute to the delay in the control loop significantly. It is well known that not only the average delay impacts the control quality, but also the variance of the delay at runtime [7]. The delay characteristic and its contribution to the overall control quality of the system can be considered during various steps in the development process. During the integration of several control applications on a distributed embedded platform, mapping and scheduling of tasks and messages should be guided by their impact on control delays and quality. Also, feedback of the delay characteristics from these steps can be used to design control laws that are customized for the actual delays in the system. Such methods that integrate traditional control design with system mapping and scheduling have been demonstrated to improve control quality compared to the traditional separated development process [8]. Integrated control and computer systems design has become an important research direction. Further, the interaction between control, computation, and communication has recently been even more emphasized in the context of cyber-physical systems [9], [10].

Several researchers have contributed towards integrated design approaches for embedded control systems. Seto et al. [11] studied uniprocessor systems that execute several control tasks. They solved the problem of optimal period assignment to each controller, with timing constraints given by two common scheduling policies: rate-monotonic and earliest-deadline-first scheduling [5]. The optimization goal is to maximize the performance of the running controllers. In that context, no delays were accounted for, until Bini and Cervin [12] extended the work to consider delays according to their proposed response-time analysis. Ben Gaid et al. [13] considered static scheduling of control signals, given one single control loop, which is closed over a communication channel, and the sampling period of the controller. The plant to be controlled, with given initial state, is assumed to be noise free. The result of the optimization is a finite sequence of control signals and start times for their transmissions over the communication channel to the actuator. Di Natale and Stankovic [14] proposed a simulated annealing-based approach that, given the application periods, constructs static schedules that minimize the jitter in distributed embedded systems with precedence and timing constraints. They did not, however, consider the impact of the schedules on the control performance. Scheduling for control systems with workload variations has been proposed by Cervin et al. [15] but in the context of uniprocessor systems.

In our previous work [16], we presented an integrated design-optimization method for distributed embedded control systems. The design parameters are the control algorithms, sampling periods, and the schedule for task executions and message transmissions (or priorities for systems with fixed-priority scheduling and CAN communication). The mapping of tasks to computation nodes has been considered given, as it has been in the other related work we have mentioned. This paper addresses mapping optimization of distributed embedded control systems. We propose a control-quality optimization heuristic based on genetic algorithms. To our knowledge, this is the first work that combines task mapping, scheduling and communication synthesis, period selection, and control-law synthesis in an integrated manner.

This paper is organized as follows. In the next section we present the system model, i.e. plant, platform and application models. In Section III, the metric which is used for control-quality will be mentioned. Section IV includes motivational examples that indicate mapping as an important design parameter for optimizing control-quality. In Section V, we will formulate the problem. Our proposed design flow will be discussed in Section VI and the experimental results will be in Section VII. Finally, the paper will be concluded in Section VIII.

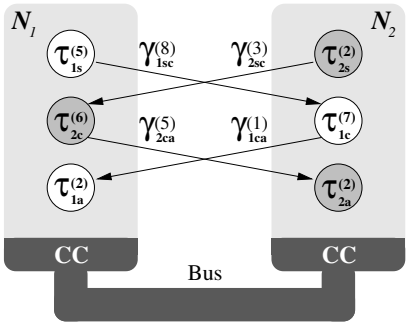


Figure 1. An example of mapping two applications on two different nodes (Communication between tasks is done through bus.)

II. System Model

A. Plant Model

Let us consider a given set of plants $\mathbf{P}$. Each plant P_i is modeled by a continuous-time linear system

$$\begin{aligned} \dot{\boldsymbol{x}}_i &= A_i \boldsymbol{x}_i + B_i \boldsymbol{u}_i + \boldsymbol{v}_i, \\ \boldsymbol{y}_i &= C_i \boldsymbol{x}_i + \boldsymbol{e}_i, \end{aligned} \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and control law, respectively. The additive plant disturbance $\boldsymbol{v}_i$ is a continuous-time white-noise process with zero mean and given covariance matrix R_{1i}. The output signal is denoted by $\boldsymbol{y}_i$ and is measured periodically by a control application. The plant outputs are sampled periodically at discrete time instants—the measurement noise $\boldsymbol{e}_i$ is a discrete-time white-noise with zero mean and covariance R_{2i}. The control signal will be updated periodically with some delays at discrete time instants and is held constant between two updates by a hold-circuit in the actuator [1].

As an example, let us consider an inverted pendulum [17] that can be modeled using Equation 1 with $A_i = \begin{bmatrix} 0 & 1 \\ -g/l_i & 0 \end{bmatrix}$, $B_i = \begin{bmatrix} 0 & g/m_i {l_i}^2 \end{bmatrix}^\mathsf{T}$, and $C_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$, where $g \approx 9.81$ m/s^2 is the gravitational constant and l_i and m_i are the length and mass of pendulum P_i respectively. The two states in $\boldsymbol{x}_i = \begin{bmatrix} \phi_i \\ \dot{\phi}_i \end{bmatrix}$ are pendulum position ϕ_i and speed $\dot{\phi}_i$. For plant disturbance and measurement noise, we have $R_{1i} = B_i B_i^\mathsf{T}$ and $R_{2i} = 0.1$, respectively.

B. Platform and Application Model

The platform in this paper consists of several computation nodes distributed in a system and a single bus for communication between nodes. Moreover, there are several control applications which control some plants in this distributed system. We indicate each computation node by $N_i \in \mathbf{N}$, where $\mathbf{N}$ denotes the set of nodes. Each plant $P_i \in \mathbf{P}$ has a corresponding control application $\Lambda_i \in \mathbf{\Lambda}$ in the system. The application set $\mathbf{\Lambda}$ can also contain a set of other applications (e.g., safety-critical applications which have strict timing constraints, or monitoring applications).

Each application is modeled as a task graph. Each task graph consists of a number of tasks and the communication between them. Thus, we model an application as a directed acyclic graph $\Lambda_i = (\mathbf{T}_i, \mathbf{\Gamma}_i)$, where $\mathbf{T}_i$ denotes the set of task and $\mathbf{\Gamma}_i \subset (\mathbf{T}_i \times \mathbf{T}_i)$ denotes the set of communications between tasks. Task j in application Λ_i is indicated by $\tau_{ij} \in \mathbf{T}_i$. A message between tasks τ_{ij} and τ_{ik} is indicated by $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i$.

A control system can typically give satisfactory performance over a range of sampling periods. This range is given by rules of thumb based on the dynamics of the plant under control [1]. The available computation and communication resources, as well as the impact of the period on mapping, schedules, and control performance, are important to consider when selecting the sampling period of each control application. Each application Λ_i executes with a period $h_i \in \mathbf{H}_i$, where $\mathbf{H}_i$ is a given set of possible periods an application can be executed with.

Let us now consider constraints related to task mapping. Each task τ_{ij} can be mapped on one of the nodes $m_{ij} \in \mathbf{M}_{ij} \subseteq \mathbf{N}$, where $\mathbf{M}_{ij}$ is the set of possible nodes that task τ_{ij} can be mapped on. An example for mapping constraints can be a temperature sensor which is placed in a specific location in a system, and the task which is going to read the temperature should be mapped on one of the nodes which the sensor is connected to. Another example is that certain tasks require application-specific instructions or hardware accelerators that are available only on some computation nodes. Thus, possible mapping nodes for each task τ_{ij}, given by $\mathbf{M}_{ij}$, are part of the design specification, based on design constraints. Hence, we have a mapping function

$$\mathsf{map}(\tau_{ij}) = m_{ij} \in \mathbf{M}_{ij},$$

that maps each τ_{ij} on one of the possible mapping nodes. The communication between tasks is mapped on the bus if and only if two communicative tasks are mapped on different nodes. Thus, the set of messages on the bus is

$$\mathbf{\Gamma}_{\text{bus}} = \{\gamma_{ijk} \in \mathbf{\Gamma}_i \mid \mathsf{map}(\tau_{ij}) \neq \mathsf{map}(\tau_{ik})\}.$$

Otherwise the communication will be done locally, and its time overhead is considered to be part of the computation time for each task.

Figure 1 shows an example of a distributed embedded control system, where we have two computation nodes N_1 and N_2 and a communication bus. We have two control applications and each of them has three tasks, where τ_{is}, τ_{ic}, and τ_{ia} indicate the sensation, computation, and actuation tasks of control application Λ_i, respectively. For communication, γ_{isc} is the communication between tasks τ_{is} and τ_{ic}, whereas γ_{ica} is between tasks τ_{ic} and τ_{ia}. If the communicative tasks are mapped on different nodes, then these communications will be done over the bus. The numbers in the parentheses determine the communication delay for a message over the bus or execution time for a task.

III. Control Quality and Synthesis

In order to measure the quality of control for the controller Λ_i for plant P_i we use the quadratic cost [1]

$$J_{\Lambda_i} = \lim_{T\to\infty} \frac{1}{T} \mathrm{E}\left\{ \int_0^T \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}^{\mathsf{T}} Q_i \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} dt \right\}. \quad (2)$$

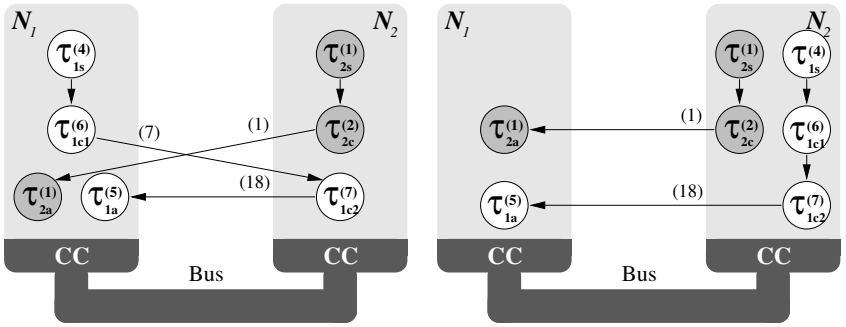
The weight matrix Q_i is given by the designer, and is a positive semi-definite matrix with weights that determine how important each of the states or control inputs are in the final control cost, relative to others ($\mathrm{E}\{\cdot\}$ denotes the expected value of a stochastic variable).

For a given sampling period h_i and a given, constant sensor–actuator delay (i.e., the time between sampling the output $\boldsymbol{y}_i$ and updating the controlled input $\boldsymbol{u}_i$), it is possible to find the control law $\boldsymbol{u}_i$ that minimizes the cost J_{Λ_i} [1]. Thus, optimal control can be achieved if the delay is constant at each periodic instance of the control application. However, the sensor–actuator delay is typically not constant at runtime because of interference experienced by other applications competing for execution on nodes or transmission over bus. The quality of a controller is degraded (its cost J_{Λ_i} is increased) if the sensor–actuator delay distribution is different from what was assumed during the control-law synthesis. In order to synthesize the controller and compute the quadratic control cost J_{Λ_i} for the constructed controller and a certain sensor–actuator delay distribution, we use MATLAB and the Jitterbug toolbox [2].

IV. Motivational Example

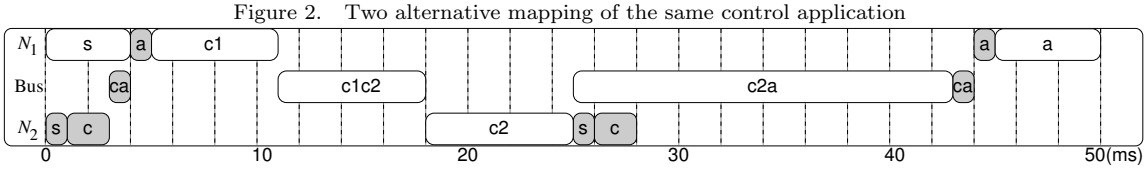
In this section, we shall motivate the need of mapping optimization in the context of integrated control and computer systems design. We present two examples: The first example shows that different task mappings lead to different delay characteristics and, consequently, different levels of control quality. We also illustrate that it is not only important to have small average delays but, in addition, the variance and jitter of the delay is a decisive parameter in the overall control quality that is achieved. The second example illustrates the need of a proper exploration of the space of possible mappings to achieve high control performance. This is illustrated by a comparison to two straightforward mapping approaches: The first straightforward approach balances the load on the computation nodes, whereas the second finds a mapping that minimizes the amount of communication on the bus.

For the examples in this section, we consider the weight matrix $Q_i = \mathrm{diag}\left(C_i^{\mathsf{T}} C_i, 0.01\right)$ for each application Λ_i. All time quantities are given in milliseconds (ms) throughout this section. Static-cyclic scheduling

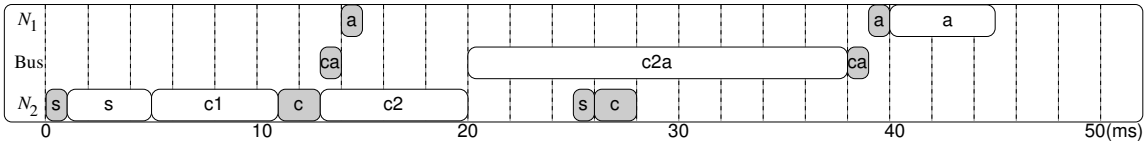


(a) Initial mapping (b) Improved mapping

Figure 2. Two alternative mapping of the same control application



(a) Schedule for initial mapping



(b) Schedule for improved mapping

Figure 3. Corresponding schedules for two alternative mapping of the same control application

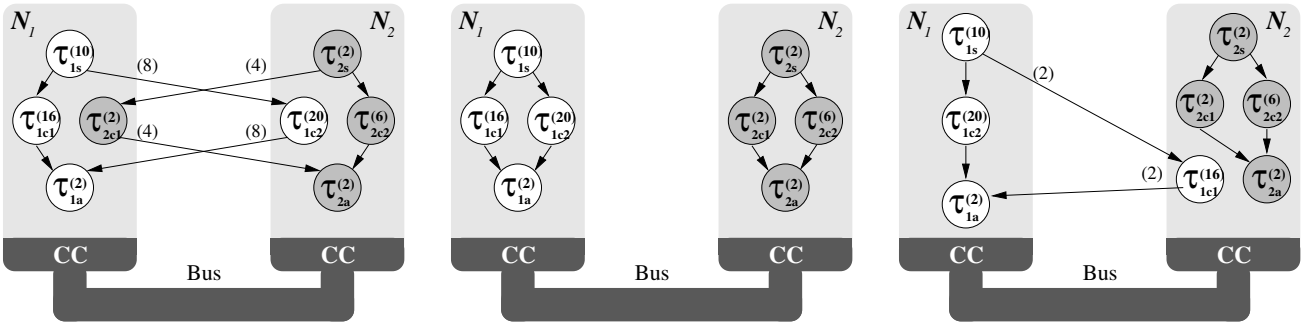
was used to schedule both tasks and messages in these examples. Control laws are synthesized assuming a delay equal to the average delay in the execution schedule. The actual important parameter related to control performance is the sampling–actuator delay distribution of the system. Thus, although we are considering static-cyclic scheduling in the examples, the conclusions of this section are also valid for priority-based scheduling and other communication protocols, which are supported by our design tool [16], [18].

A. Example 1

In this example, we will use two control applications. There are two computation nodes and each task can be mapped on each of them. The first control application consists of four tasks and the second control application consists of three tasks. The period for the first control application is equal to 50 ms, whereas the period for the second control application is 25 ms. The hyperperiod is thus 50 ms.

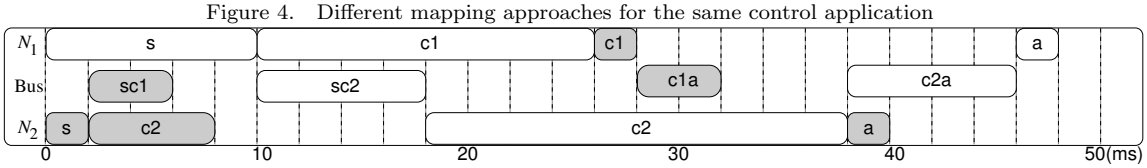
One possible mapping of the tasks is shown in Figure 2(a). For this mapping, we have run our scheduling and synthesis tool [16] to obtain schedules, control laws, and periods for the two control applications. Figure 3(a) shows the obtained schedule. The schedule is shown in three rows for node N_1, the bus, and node N_2, respectively. Each box depicts either a task execution (if it is on one of the nodes) or a message transmission (if it is on the bus). Boxes in white correspond to application Λ_1, whereas the gray boxes show execution of tasks and transmission of messages for application Λ_2. Each box has a label which specifies the corresponding task or message. For instance, the box which is labeled $c1c2$ in Figure 3(a) is in white color and on the bus; therefore, it is the message between task τ_{1c1} and task τ_{1c2} in application Λ_1. Control laws have been synthesized for the two applications, assuming a constant runtime delay between sampling–actuation delay. This delay is chosen to be the average delay given by the schedule. The two controllers have been evaluated in Jitterbug [2], considering the actual delay distribution given by the schedule. This leads to a total control cost of 4.5, where the first and second control applications have costs 3.4 and 1.1, respectively.

Now, let us change the mapping according to Figure 2(b). The corresponding schedule for the new mapping is shown is Figure 3(b). As a result of change in mapping and consequently schedule, the control cost is decreased to 3.8. If we consider each control application separately, we will see that for the first control application, the cost was decreased from 3.4 to 3.0, and it is because the sensor–actuator delay was decreased in the second mapping. However, if we consider the second control application, the average sensor–actuator delay was increased from 12.5 ms to 15 ms, but the jitter was decreased from 15 ms to zero (i.e., we achieve a constant-delay execution of the controller). The control

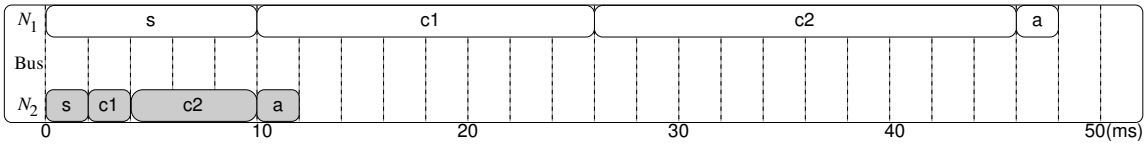


(a) Load balancing (b) Communication minimization (c) Control-quality aware mapping

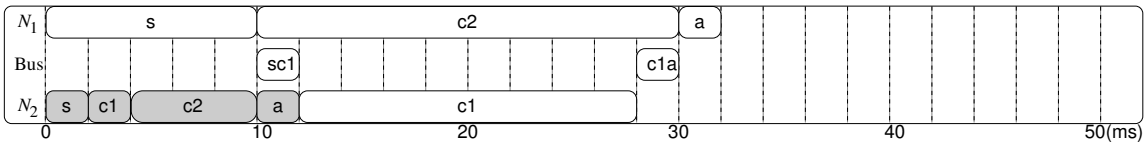
Figure 4. Different mapping approaches for the same control application



(a) Schedule for load balancing heuristic



(b) Schedule for communication minimization heuristic



(c) Schedule for the control-quality aware mapping

Figure 5. Corresponding schedules for different mapping approaches

cost for the second control application is decreased from 1.1 to 0.8, which is an improvement of 27%. This example illustrates the fact that not only the average delay is a key factor in the control quality, but also the variance of the delay at runtime (i.e., a larger but less varying delay can be better than a solution with a small average sensor–actuator delay and with large amounts of jitter).

B. Example 2

Let us consider two control applications, each comprising four tasks: τ_{is}, τ_{ic1}, τ_{ic2}, and τ_{ia}. There are two computation nodes and each task can be mapped on any of the two nodes. The period for both control applications is equal to 50 ms. We shall discuss three different mappings and discuss their quality in terms of control quality.

First, let us consider a task mapping that balances the computational load on the two nodes. The intuition here is that such a task mapping would yield good control quality, as the load is distributed evenly on the computation nodes in the system. Thus, we have tasks τ_{1s}, τ_{1a}, τ_{1c1}, and τ_{2c1} mapped on N_1, whereas tasks τ_{1c2}, τ_{2s}, τ_{2c2}, and τ_{2a} are mapped on N_2. The mapping of tasks, including data dependencies as well as computation and communication times, are shown in Figure 4(a). The constructed schedule for this mapping is shown in Figure 5(a). In this example, the sum of execution time on each node is equal to 15 ms. Considering that all tasks run with the same period of 50 ms, we observe that the computational load is equal on the two nodes. For this mapping, and the constructed schedule and control laws [16], we computed the total control cost to 6.0.

Next, let us consider a task mapping that minimized the amount of communication on the bus. The intuition here is that less communication on the bus leads to smaller delays in the control loop, thus giving possibilities to construct good controllers. Figure 4(b) shows the new mapping of the two control applications. Since all tasks of controller one are mapped on N_1 and all tasks of controller two are mapped on N_2, we have the least possible amount of communication. Figure 5(b) shows the corresponding schedule for this example. We have constructed a schedule and control laws for this new mapping [16], and obtained a solution with a control cost of 4.6, which is a better solution in terms of control

quality compared to the previous mapping with load balancing (i.e., a decrease of 1.4 in the total control cost).

Finally, let us consider that tasks τ_{1s}, τ_{1c2}, and τ_{1s} are mapped on node N_1, whereas tasks τ_{1c1}, τ_{2s}, τ_{2c1}, τ_{2c2}, and τ_{2a} are mapped on N_2. The corresponding mapping and schedule are shown in Figures 4(c) and 5(c), respectively. We computed the total control cost for this final solution to 3.5, indicating 42% an 24% improvements in control quality compared to load balancing and communication minimization, respectively.

Thus, we can conclude that neither load balancing nor communication minimization will necessarily lead to a high control performance, though in general they are better than a purely random mapping. However, a good mapping algorithm should consider both balancing the load and minimizing the communication at the same time, as well as the direct relation between mapping, scheduling, controller synthesis, and quality of control. Let us again consider our three mappings discussed in this section. For the first mapping, the load was perfectly balanced and communication overhead was the main problem. In contrast, for the second mapping, communication was removed but the load was not balanced at all. In the last mapping, the load was less balanced in comparison with the first mapping, and the communication was more than the second mapping. This trade-off lead to the best control quality out of the mappings that we have studied in this section.

Finally, let us assume $35 \in \mathbf{H}_1, \mathbf{H}_2$. As Figure 5(c) shows, we can decrease the period of the schedule from 50 ms to 35 ms for the last mapping. We have modified the solution for the last mapping accordingly and obtained a design solution with the control cost 2.6. Hence, we conclude that the mapping, scheduling, period selection, as well as the actual control laws all impact the final control quality that is achieved by the system.

V. Problem Formulation

The inputs to the mapping problem are

- a set of plants $\mathbf{P}$ to be controlled,
- a set of applications $\mathbf{\Lambda}$ among which a subset of them are the controllers for the plants,
- a set of computation nodes $\mathbf{N}$ connected to a bus,
- a set of available sampling periods $\mathbf{H}_i$ for each control application Λ_i and the release period h_j for the other applications Λ_j,
- a set of nodes $\mathbf{M}_{ij}$ that each task task $\tau_{ij} \in \mathbf{T}_i$ can be mapped to,
- deadlines of a subset of the tasks (possibly no deadlines),
- a scheduling policy for the tasks and messages, and
- execution–time distributions of the tasks and communication times of messages.

The outputs of the tool are a mapping of each task $\tau_{ij} \in \mathbf{T}_i$, controller periods $h_i \in \mathbf{H}_i$, schedule table for the nodes and the bus (or priorities if fixed-priority scheduling and CAN communication is used), and the control law $\boldsymbol{u}_i$ for each plant $P_i \in \mathbf{P}$. The outputs related to the controller synthesis are the period $h_i \in \mathbf{H}_i$ and the control law $\boldsymbol{u}_i$ for each plant P_i. The outputs for static-cyclic scheduling is a schedule table with start times of the job executions and the communications on the bus. It must be guaranteed that task deadlines are met at runtime. As mentioned before, there exists a control application $\Lambda_i \in \mathbf{\Lambda}$ corresponding each plant $P_i \in \mathbf{P}$ and the final cost is the weighted sum of the individual control costs J_{Λ_i} (Equation 2) of all control applications $\Lambda_i \in \mathbf{\Lambda}$. Hence, the cost function to be minimized is

$$\sum_{P_i \in \mathbf{P}} w_{\Lambda_i} J_{\Lambda_i}, \tag{3}$$

where the weights w_{Λ_i} are determined by designer.

VI. Mapping Approach

In this section, first we discuss the overall structure of our tool. After that, we elaborate on effects of mapping on control quality. In the last part of this section, we present the genetic algorithm-based mapping algorithm that is implemented in our design tool.

A. Overall Solution

Figure 6 illustrates the overall structure of our tool. This flowchart shows iterations of a genetic algorithm over four steps until the the algorithm is terminated by the stopping condition. The genetic algorithm-based mapping approach considers, in each iteration, a population of members, where each member is a solution candidate. At each iteration, a population is evaluated in terms of its control quality, followed by a modification of the population before the next iteration. In the first step, for each member of the current population, we have a task mapping and periods for the control applications, which all satisfy the imposed constraints related to mapping and period assignment. Thus, the vector that indicates each member in the genetic algorithm includes both mapping of tasks and periods of controllers. In our previous work [16], we only considered periods; thus, a member was characterized only by periods, as the task mapping was assumed to be given as an input. Hence, the genetic algorithm chooses vectors

$$v = (h_1, ..., h_{|\mathbf{\Lambda}|}, m_{11}, ..., m_{|\mathbf{\Lambda}||\mathbf{T}_\mathbf{\Lambda}|}) \in \prod_{i=1}^{|\mathbf{\Lambda}|} \mathbf{H}_i \times \prod_{i=1}^{|\mathbf{\Lambda}|} \prod_{j=1}^{|\mathbf{T}_i|} \mathbf{M}_{ij},$$

where $h_i \in \mathbf{H}_i$ is the selected period for application Λ_i and $m_{ij} \in \mathbf{M}_{ij}$ is the node which task $\tau_{ij} \in \mathbf{T}_i$ of application Λ_i is mapped on.

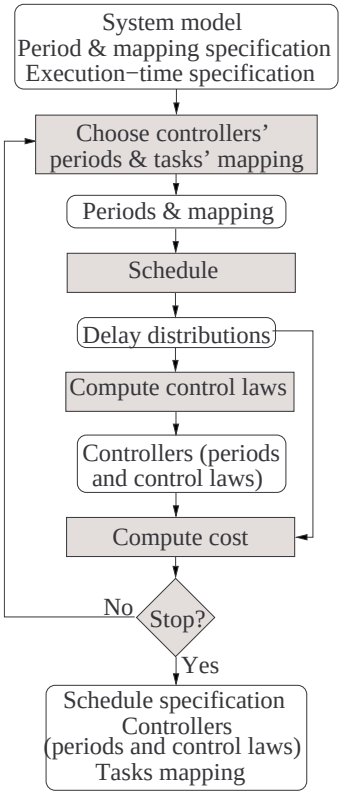


Figure 6. Overall approach

In the second step, the tool schedules the tasks and messages according to the selected periods and the given data dependencies. Also, if other hard real-time applications co-exist with the control applications, we have timing constraints given by the hard deadlines. In the last two steps, the constructed schedule, which corresponds to a certain mapping and period assignment v, is evaluated. First, for each application, a controller is constructed for the given period and the average sensor–actuator delay in the constructed schedule. After this control-law synthesis, we compute the control cost J_v for each mapping and period assignment v of the population by using the Jitterbug toolbox [2] and providing as inputs the constructed controller and the delay distribution given by the schedule.

B. How does mapping affect control quality?

Having introduced the overall approach, let us proceed with a discussion about how mapping affects control quality. As it is visualized in Figure 7, the mapping process has a direct effect on both period assignment and scheduling. In other words, a good mapping can lead to a better schedule and period assignment (from the point of view of control quality) and consequently higher control performance which cannot be achieved with another mapping. Further, the assigned periods affect the schedule.

The selected periods together with the average sensor–actuator delay extracted from schedule are used to construct the controller. Finally, the sensor–actuator delay distribution extracted from schedule will be used to compute control cost for constructed controllers. Thus, as we have already seen in the second motivational example, mapping indirectly have noticeable impacts on control cost. In the context of Figure 7, the period and mapping exploration is performed according to our proposed approach in this paper. The scheduling is performed by our existing design tool [16] based on a list-scheduling based heuristic. The synthesis of the control law, as well as the computation of the control cost is done in MATLAB and Jitterbug [2].

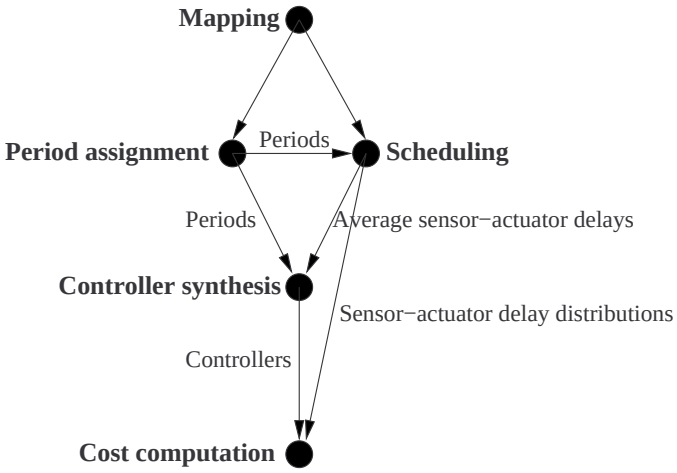


Figure 7. Visualization of mapping impact on quality of control

C. Genetic Algorithm-Based Heuristic

Mapping of tasks over several computation nodes is a combinatorial NP-complete problem and exhaustive search techniques are not feasible for this kind of problems since the number of solutions will grow exponentially with problem size and an optimal algorithm has unaffordable runtime for reasonably large problem sizes. We have developed a genetic algorithm-based heuristic [19] for the exploration of mappings and periods.

The size of the population is fixed and remains constant throughout all iterations of the genetic algorithm. The size depends on the size of the application set $\mathbf{\Lambda}$, maximum size of mapping sets $\mathbf{M}_{ij}$ of all tasks, and maximum size of period sets $\mathbf{H}_i$ of all applications.

For the initial population in our genetic algorithm, we generate a set of vectors randomly according to the mapping constraints. In each iteration (also referred to as generation) and for each member, we construct a schedule and control laws, followed by computation of the control cost for that member. Next, we select a number of members that shall survive and move to the next generation. This selection is based on their control cost—the smaller the control cost is, the higher is the

Table I
COMPARISON BETWEEN OUR PROPOSED APPROACH AND LOAD-BALANCING HEURISTIC

Number of tasks	Percentage of tests LB could find a solution	Improvement $\left(\frac{J_{LB}-J_{GA}}{J_{LB}}\right) \times 100$
10	100%	10%
15	95%	26%
20	77%	40%
25	35%	56%
30	16%	54%
35	1.6% (1 case)	33%
40	2.8% (2 cases)	92%

Table II
COMPARISON BETWEEN OUR PROPOSED APPROACH AND COMMUNICATION MINIMIZATION HEURISTIC

Number of tasks	Percentage of tests CM could find a solution	Improvement $\left(\frac{J_{CM}-J_{GA}}{J_{CM}}\right) \times 100$
10	100%	18%
15	90%	22%
20	63%	25%
25	60%	27%
30	62%	25%
35	53%	30%
40	52%	30%

probability of being included in the next generation. Moreover, crossover and mutation are used to create offsprings that are included in the next population, together with the already selected members from the previous generation.

For the mutation, a task's mapping or a controller's period is changed according to mapping and period constraints randomly (i.e., $h_i^{\text{new}} \in \mathbf{H}_i$, $m_{ij}^{\text{new}} \in \mathbf{M}_{ij}$). The mutation probability is equal to 0.15 and will decrease when the number of applications increases. The reason behind this is that the probability of a member to be modified grows with number of tasks.

The crossover is done based on several crossover points and preserves the mapping constraints. As an example, let us assume we want to do the single-point crossover for two members v and v':

$$v = (h_1, h_2, ..., h_{|\mathbf{\Lambda}|}, m_{11}, ..., m_{|\mathbf{\Lambda}||\mathbf{T_\Lambda}|})$$
$$v' = (h'_1, h'_2, ..., h'_{|\mathbf{\Lambda}|}, m'_{11}, ..., m'_{|\mathbf{\Lambda}||\mathbf{T_\Lambda}|}).$$

For simplicity of presentation, let us assume that the crossover point is between h_1 and h_2 (h'_1 and h'_2). Then we change the place of h_1 with h'_1, leading to two offsprings:

$$v = (h'_1\ |, h_1, ..., h_{|\mathbf{\Lambda}|}, m_{11}, ..., m_{|\mathbf{\Lambda}||\mathbf{T_\Lambda}|})$$
$$v' = (h_1\ |, h'_2, ..., h'_{|\mathbf{\Lambda}|}, m'_{11}, ..., m'_{|\mathbf{\Lambda}||\mathbf{T_\Lambda}|}).$$

The crossover for the part related to mapping is done in a similar manner by first generating a crossover point randomly. Thus, our crossover implementation uses multiple crossover points to generate offsprings with different periods and mappings. The probability for crossover is equal to 0.4 and has been decided based on experiments.

The average distance between members in the population was used as a stopping condition to measure the homogeneity of the population. The distance between two members is defined as the number of individual components of the vectors that are different. If the average distance between members of a population is less than *bound* $\in [0.04, 0.28]$ of the length of each member's vector for more than 10 consequent iterations, and the algorithm did not find a better solution for 50 generations, then the design-space exploration terminates and the best solution in terms of control quality is provided.

VII. EXPERIMENTAL RESULTS

In order to study the improvements that can be achieved by our tool, several experiments were performed. The results of the experiments are organized in the following manner. In the first part we will compare our genetic algorithm-based approach with an exhaustive search of all possible mappings. Further, we compare our heuristic with two straightforward mapping approaches that balance the load and minimize bus communication, respectively. In the last part of this section, we discuss the runtime of our proposed approach.

A. Comparison with optimal mapping

In this section, we compare our genetic algorithm-based approach (GA) with optimal mapping (OM) with regard to runtime and control quality. Due to the design-space complexity, we could not afford to run OM for test cases with more than 10 tasks. For some cases with 15 tasks, for example, OM took more than 30 hours to explore the set of all possible mappings. By studying the relative difference between the control costs obtained by the GA and OM approaches, we observed that our GA approach is 2.8% away from the minimum cost obtained by OM.

B. Comparison with straightforward approaches

We have defined two straightforward design approaches for mapping: load balancing (LB) and communication minimization (CM). These shall serve as baselines for comparison.

1) Load-balancing (LB): The load-balancing approach is a greedy algorithm and tries to balance the load on the computation nodes as much as possible. This heuristic finds the task which has the least possible options for mapping (i.e., the most constrained task in terms of the number of possible nodes it can be mapped on). This task is then mapped on the node that has currently the least amount of load and that satisfies the mapping constraints for that task. Then the heuristic proceeds in a similar manner for the set

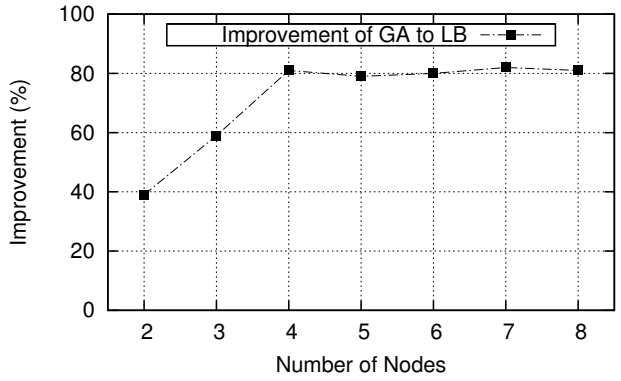


Figure 8. Improvement of our method compared to the load-balancing approach for 25 tasks running on varying number of computation nodes

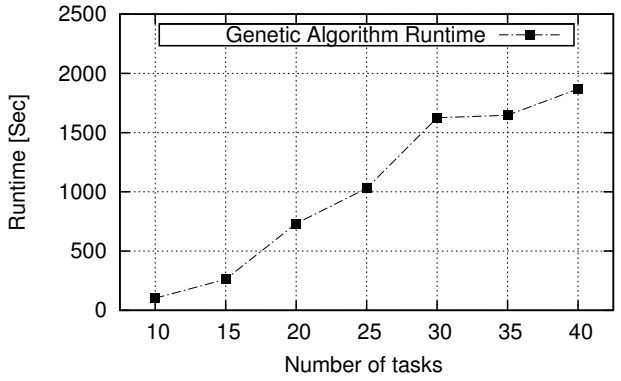


Figure 9. Runtime of our genetic algorithm-based mapping heuristic

of unmapped tasks. The heuristic terminates when all tasks are mapped.

2) Communication minimization (CM): The communication-minimization approach minimizes the communication of tasks over the bus. This heuristic considers each control application separately and minimizes the communication of each control application by mapping tasks which need to communicate with each other, on the same node, if it is possible with regard to mapping constraints.

For the evaluation of our genetic algorithm-based approach we have used 265 benchmarks with varying number of plants and computation nodes. The number of plants is between 2 to 8 and taken from a set of inverted pendulums, ball and beam processes, DC servos and harmonic oscillators [1]. Such benchmarks are representatives of realistic control problems and are used extensively for experimental evaluation. For each plant, we generated a control application with 5 tasks and with different structures related to data dependencies. The number of allowed mapping nodes for each task can be as large as the number of computation nodes. For comparison between the genetic algorithm-based approach (GA) and a straightforward (SF), we are interested in the relative cost improvement $\frac{J_{SF}-J_{GA}}{J_{SF}}$, where J_{SF} is the cost obtained by either of the two straightforward approaches LB or CM, and J_{GA} is the cost of the design solution obtained by our proposed approach based on genetic algorithm.

The results are shown in Table I and Table II. There are two columns for each of the two straightforward approaches (LB and CM). The first column indicates the percentage of the cases in which straighforward approach could find a valid solution (i.e., schedulable and stable). The second column indicates the improvement in control quality achieved by our approach compared to the corresponding straightforward approach. It is important to note that our GA could find valid solutions for all the benchmarks.

For the LB heuristic we can observe a clear trend in the experimental results. As the size and complexity of the test cases increase, the number of benchmarks that can be handled by the LB approach decreases. Besides that, the improvement of our approach is increasing when the size and complexity of the benchmarks increase. For the CM heuristic, in general, the percentage of the benchmarks for which CM could find a solution is decreasing with the increase in problem size and complexity. We also observe a notable improvement in control quality (e.g., 30% for systems with 40 tasks).

C. Varying number of nodes for a fixed number of controllers

To further evaluate our proposed optimization approach, we performed experiments on a system with a fixed number of 25 tasks, but with platforms with varying number of computation nodes. The number of computation nodes vary between 2 and 8. The results are shown in the Figure 8. On the horizontal axis, we show the number of nodes in the system, whereas on the vertical axis we show the relative improvement in control quality compared to a solution obtained by the load-balancing heuristic. When the number of nodes are increased, the improvement increases. This is due to the fact that if we have several nodes then load balancing is not the only issue to consider during optimization and our optimization approach finds other combinations of mapping, scheduling, periods, and controllers that are superior in terms of their control quality. For systems with 5 nodes or larger, the improvement saturates. This is because the amount of computation power in the platform is more than what is needed to achieve high control quality for the system with 25 tasks.

D. Runtime of our proposed approach

We measured the runtime of our mapping approach, running on a PC with a quad-core CPU running at 2.83

GHz with 8 GB of RAM and Linux. The average runtime of our proposed approach, based on number of tasks is shown in the Figure 9. As can be observed, our design tool can be used to construct a design solution (mapping, schedule, and controllers) with high control quality for large systems of 40 tasks in less than 32 minutes.

VIII. Conclusions

In this paper, we have demonstrated that task mapping has an important and complex relation to the quality of control for distributed embedded systems running multiple feedback-control loops. Efficient heuristics are needed to handle the complex design space related to mapping, scheduling, period selection, and control synthesis. We proposed a control-quality driven mapping approach for distributed embedded control systems. Our experimental results show that our optimization approach, which considers mapping, scheduling, period selection, and control synthesis in an integrated manner, is essential to construct distributed embedded control systems with high quality.

References

[1] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems*, 3rd ed. Prentice Hall, 1997.

[2] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *Proceedings of the* 41^{st} *IEEE Conference on Decision and Control*, 2002, pp. 1319–1324.

[3] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. E. Årzén, "How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime," *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, 2003.

[4] H. Kopetz, *Real-Time Systems—Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 47–61, 1973.

[6] R. Bosch GmbH, *CAN Specification Version 2.0*, 1991.

[7] B. Wittenmark, J. Nilsson, and M. Törngren, "Timing problems in real-time control systems," in *Proceedings of the American Control Conference*, 1995, pp. 2000–2004.

[8] K. E. Årzén, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proceedings of the* 39^{th} *IEEE Conference on Decision and Control*, 2000, pp. 4865–4870.

[9] E. A. Lee, "Computing needs time," *Communications of the ACM*, vol. 52, no. 5, pp. 70–79, 2009.

[10] ——, "Cyber physical systems: Design challenges," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-8, 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html

[11] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proceedings of the* 17^{th} *IEEE Real-Time Systems Symposium*, 1996, pp. 13–21.

[12] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*, 2008, pp. 291–300.

[13] M. M. Ben Gaid, A. Cela, and Y. Hamam, "Optimal integrated control and scheduling of networked control systems with communication constraints: Application to a car suspension system," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 4, pp. 776–787, 2006.

[14] M. Di Natale and J. A. Stankovic, "Scheduling distributed real-time tasks with minimum jitter," *IEEE Transactions on Computers*, vol. 49, no. 4, pp. 303–316, 2000.

[15] A. Cervin, J. Eker, B. Bernhardsson, and K. E. Årzén, "Feedback–feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1–2, pp. 25–53, 2002.

[16] S. Samii, A. Cervin, P. Eles, and Z. Peng, "Integrated scheduling and synthesis of control applications on distributed embedded systems," in *Proceedings of the Design, Automation and Test in Europe Conference*, 2009, pp. 57–62.

[17] M. Karam and M. Zohdy, "Modeling a simple inverted pendulum using a model-based dynamic recurrent neural network," in *SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory*, 2005, pp. 78–82.

[18] S. Samii, P. Eles, Z. Peng, and A. Cervin, "Design optimization and synthesis of FlexRay parameters for embedded control applications," in *Proceedings of the* 6^{th} *IEEE International Symposium on Electronic Design, Test and Application*, 2011, pp. 66–71.

[19] C. R. Reeves, Ed., *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.