Simulation-driven optimization of real-time control tasks

Matteo Morelli, Yasmina Seddik
CEA, LIST, Laboratory of Model-Driven Engineering for Embedded Systems
Email: {matteo.morelli, yasmina.seddik}@cea.fr

Marco Di Natale
Scuola Superiore S. Anna
Email: marco@sssup.it

Chokri Mraidha, Sara Tucci-Piergiovanni
CEA, LIST, Laboratory of Model-Driven Engineering for Embedded Systems
Email: {chokri.mraidha, sara.tucci}@cea.fr

***Abstract*—In this paper we define a simulation-driven process to improve the design of real-time control systems. The process aims at exploring the interplay betwen control performance and real-time behavior of control tasks. The traditional design flows based on the definition of implicit tasks deadlines on control functions are extended to include the exploration of relaxed deadlines and order of execution constraints. Relaxed deadlines, coupled with an optimization approach to find feasible task sets, allow the exploration and evaluation of different task implementations. The definition of relaxed deadlines and the evaluation of task implementations is performed using the T-Res (Time and Resource) scheduling simulation framework [21] under Simulink. The problem is defined as a quadratic optimization problem using a tight upper bound formulation of the task response times. The application of the method to a quadcopter case study shows how the consideration of the control performance in the definition of the timing parameters of interest can lead to an improved design.**

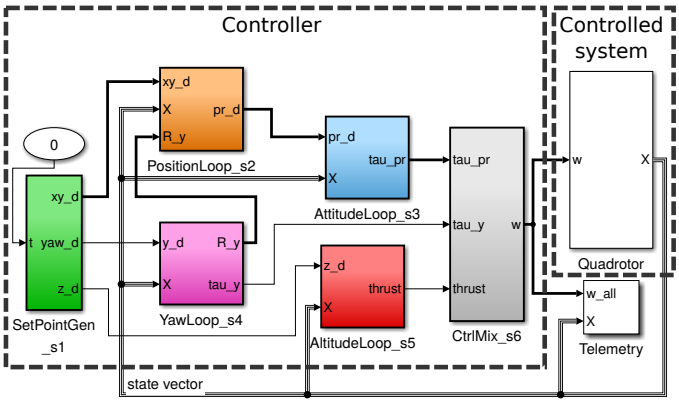


Fig. 1: Simulink model of the quadcopter flight-control scheme (adapted from [22])

I. Introduction

The design of real-time control systems is typically performed in two steps. First, the control system is designed as a graph of functional blocks activated at a given rate (sampling period). Then, the software implementation is designed as a set of real-time tasks in charge of executing the functional code. The sampling periods of the functions, determined in the control design phase, become timing constraints in the software implementation phase, and deadlines are often assumed as implicit, meaning that each task instance must complete before the next activation. The software designer must define a feasible task set where each task meets its deadline.

In real-time control systems, however, the interplay of the control performance, timing constraints and scheduling effects can be somewhat subtle and the traditional design flow may be ineffective and result in deadlines tighter than necessary. To clarify this point let us introduce a case study of a quadcopter system adapted from [22] and developed in Simulink. The Simulink language by Mathworks is often used in the industry to model the (continuous time) set of differential equations defining the dynamics of the controlled system (or Plant) and the (discrete-time) model of the controller functionality, to be implemented in software. The Simulink model of the quadcopter system is shown in Figure 1.

The Simulink model of the controller functionality contains a set of subsystem blocks, representing the processing to be performed at each iteration of the control loop. In Simulink, these computations are specified as executing in logical time with a given period and under the synchronous assumption, that is, each reaction must complete before the next event in the system.

In our sample quadcopter system, the physical system is represented by the white box on the right (`Quadrotor`). The other subsystems represent (left to right): the generator of the setpoints of the desired trajectory, the position, yaw, altitude and attitude control loops and the merger of the control actions to drive the actuators. They have different execution rates, that range from 10 Hz for reading (generating) set-points to 50 Hz for controlling the vehicle attitude. The quadcopter model is used as a running example and case study throughout the paper.

When the model is implemented in code, two code functions are generated for each subsystem: one performs its initialization at startup, and another (called *Step*) dynamically performs the update of the subsystem outputs and state at each periodic execution of the control loop.

The simulation results retain their validity upon condition that the implementation preserves the execution semantics. The functions implementing the subsystems must be activated at a rate matching the logical time specification of the model, they must execute in an order consistent with the input-output dependencies of the model and the code implementation of each subsystem must complete before the next trigger event.

The first constraint is taken into account by executing the subsystem functions in tasks activated with a period that

corresponds to the subsystem period. The order of execution is satisfied by executing the predecessor subsystem code before the successor (when running in the same task) or by executing the predecessor in a task with higher priority. Finally, the last requirement translates into deadlines that apply to the tasks running the control code (each task must have a deadline smaller than or equal to its period).

When an implementation according to these rules is not possible because the resulting task set would not be schedulable, the designer is forced to explore other options, including possibly relaxing deadlines and allowing for additional delays in the control functions.

Previous work [26] in the context of real-time task design optimization has focused on the assumption of implicit deadlines, performing the assignment of the subsystem functions to tasks in such a way that a timing metric (typically latency-dependent) is optimized. However, a task configuration that is only optimized with respect to timing metrics can easily be inefficient.

A. Aim of this work

In this work we define a simulation-driven process intended to improve the design flow of real-time control systems. The defined process aims at exploring the interplay between control performance and real-time behavior for a better design.

To represent computation and communication delays in Simulink, a possible solution is provided by the TrueTime toolbox [31]. TrueTime allows to model multi-task real-time kernels and networks in Simulink models of networked embedded control systems, and study the impact of lateness and deadline misses on controls. In TrueTime, the task model and the control logic are coded as a set of MATLAB functions. As a result, the integration of TrueTime with existing Simulink control toolboxes and models is difficult and requires substantial rewriting.

T-Res [21] is a novel Simulink-based co-simulation framework that enables the evaluation of the impact of latency and jitter on the performance of control functions. in T-Res the scheduler and task model are explicitly represented by Simulink blocks, interacting with the blocks of the functional model. Accordingly, T-Res integrates seamlessly with existing Simulink models and toolboxes, with only limited and localized changes.

Our design process is in three stages. In the first stage, relaxed deadlines are obtained on an estimate of control performance with respect to latency. Simulations are performed on the Simulink model of the controller by applying increasing delays on the output ports of each subsystem, one subsystem at a time. The simulation runs aim at estimating the maximum delay that can be applied to each subsystem in isolation before the control performance deteriorates in an unacceptable way.

In the second stage, the mapping (or implementation) of subsystems onto periodic tasks scheduled by priority is computed using an optimization formulation. To find an effective mapping (functions into tasks and priority assignment to tasks) we encode the problem as a mathematical optimization process. The problem is quadratic, but formulated as a Mixed-Integer Linear Programming (MILP), encoding the response time formulation that is obtained from schedulability analysis theory as a set of linear constraints. Different metrics are tried and evaluated according to the simulated performance results. Different mappings considering both minimal deadlines and relaxed deadlines are considered.

In the third stage, the task mappings obtained as optimal solutions by the MILP solver are evaluated using the T-Res framework, to estimate the control performance and compare the effectiveness of different metric functions and approaches.

B. Related work

Several approaches have been proposed in the literature to optimize control performance under schedulability constraints. In [3], tasks periods are selected to optimize control performance on a monoprocessor system with the EDF unit utilization bound. The work is extended in [5] by considering the impact of control delays on performance. In [11], task periods are dynamically adapted to optimize the performance of controls within the schedulability boundary for a uniprocessor platform.

The study of the optimal priority-based network scheduling with respect to controls performance can be found in [9]. In [6], a genetic algorithm is used to select task periods and scheduling to optimize the control performance on a distributed platform. An extension of the algorithm applies to FlexRay networks [10]. In [7], task sampling periods and bus scheduling are synthesized for distributed time-triggered (FlexRay-based) platforms. The sampling periods are enumerated from a set of possible values and the problem is solved by integer linear programming (ILP). Most recently, in [12], both the control quality (in the average-case) and the control robustness (in the worst-case) are considered in the exploration of sampling periods, system scheduling and control synthesis. The controller periods are optimized by using the coordinated search method combined with the direct search method.

Most of related works define design processes based on an analytical formulation of control and real-time scheduling co-design problem. They are tied on specific control-oriented performance metrics, usually from the optimal-control theory. Our approach couples MILP with simulation. In general, simulation allows to capture effects like cache faults, preemption of message transmission attempts, task migration delays, finite copy time of messages between driver and adapter levels, etc., that are difficult to model analytically. We synthesize SW implementations of controls by solving multiple MILP problems with different criteria and constraints related to real-time performances. Then, we use simulation to evaluate the effects of the different SW implementations on the control performances, and select the implementation having the smallest impact.

The paper is organized as follows. In Section II, we describe the system model. In Section III, we describe the simulation framework and the computation of the relaxed deadlines. In Section IV we describe the optimization model, in particular the considered constraints and the optimized metrics. In Section V the proposed approach is run on the quadcopter case study. Finally, we draw some conclusions in Section VI.

II. System model

We consider systems with a single CPU, on which a set of n functions $\mathcal{F} = \{f_1, f_2, \dots f_n\}$ obtained as the code implementation of Simulink models must be executed. Each function f_i is the code implementation of a subsystem s_i (for convenience we assign the same index to a subsystem and its function) and has a worst-case execution time (WCET) $C_i > 0$ and a period $P_i > 0$ (matching the period of the subsystem they realize), $i = 1, \dots, n$.

Simulink subsystems communicate by exchanging signals. In the code implementation these signals are realized as (possibly shared) communication variables. Each signal has a sender and a destination subsystem/function. In the model simulation, it is transmitted in zero logical time. Signals dependencies correspond to order of execution constraints when the outputs of the receiver subsystem are computed as a function of its input values (as opposed to its state only). Functions and signals can thus be represented as a directed acyclic graph in which nodes are functions and edges are signals. We use the notation $f_i \rightarrow f_j$ to indicate that f_j must execute after f_i according to the transitive closure of the order of execution constraints. Sink functions are those functions that do not have successors in the graph, and source functions have at least one signal that is not received from any predecessor (meaning that they process information coming from sensors, or external input.)

We define the set of all graph paths $\mathcal{P} = \{p_1, p_2, \dots p_q\}$ from a source to a sink.

A mapping is determined by:

- A partition of functions $f_1, \dots, f_n$ on tasks $T_1, \dots, T_h$, $h \leq n$: each function is assigned to exactly one task. The functions are called by the task in order.
- An order of execution of functions within a task.
- The priority level π_i that is assigned to each task τ_i. Priorities define a total order on tasks. By extension, the priority level of any function executed by τ_i is also π_i.

Inter-task communications are performed through finite length buffers with suitable size [28]. A function samples its inputs (resp. produces its output) at execution start (resp. end).

A. Response time analysis

The performance of the code implementation of the control algorithms depends on their latencies and jitter, which are in turn dependent on the response times of the functions. In order to estimate these response times, we make use of established and recent results on schedulability analysis.

1) Deadlines within the interarrival times: When response times are guaranteed (by construction or by adding constraints) to be less than or equal to periods, the worst case response time of each function can be computed in correspondence to its critical instant, when the task in which it executes is activated at the same time with all other higher priority tasks. Analytically, the worst-case response time r_i of f_i can be computed as (from [23], a straightforward extension of the task-based formulation for periodic tasks in [1]):

$$r_i = C_i + \sum_{j \in prec(i)} C_j + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j \qquad (1)$$

where $prec(i)$ is the set of functions that are in the same task as f_i but invoked before it, and $hp(i)$ indicates the set of all functions executed by tasks with priority higher than the task implementing f_i.

2) Deadlines larger than the interarrival times: When the system also allows for functions response times that are larger than periods, that is, when a task may be activated again when it is still awaiting its completion, the previous formula (1) may be optimistic and the exact formulation (as in [24]) requires considering all the task activations in the *busy period* of level π_i. The exact formulation becomes very difficult to encode in a formal linear or convex optimization formulation and it is therefore discarded in favor of a recent upper bound $\overline{r}_i \geq r_i$, as defined in [2].

$$\overline{r}_i = \frac{C_i + \sum_{j \in prec(i)} C_j + \sum_{j \in hp(i)} C_j(1 - U_j) - \gamma_i}{1 - \sum_{j \in hp(i)} U_j} \qquad (2)$$

where $U_j = C_j / P_j$ is the utilization of function f_j, and γ_i is defined as:

$$\gamma_i = \sum_{j,k \in hp(i), j<k} \min\{P_j, P_k\} U_j U_k \qquad (3)$$

III. Time and Resource Aware Simulation in Simulink

The evaluation of the performance of the control functions by simulation, considering the functions and tasks execution times and scheduling delays is performed in Simulink using the T-Res [21] co-simulation framework. T-Res is designed according to object-oriented design patterns to provide the integration of the Simulink simulation engine with a real-time scheduling simulator that complies with a simple event API.

T-Res is implemented as a set of custom blocks representing the scheduler and the tasks that execute at all major steps. Every time a major step occurs, the block implementing the scheduler within the operating system kernel is invoked and processes (if there is any) the task arrival events. Next, it queries the scheduling simulator to determine future events (execution completions and context switches) and uses the Simulink API to define major steps in the simulation at all the points in time in which a task scheduling event occurs.

Tasks execute according to a model of (time-consuming) computations. In T-Res the execution of a task proceeds according to units called *segments* that correspond to the execution of the functions that are called by the task main code and implement the subsystems in the model. Each segment is identified by an execution time and all segments execute in a sequence. Segments represent the execution of Simulink subsystems and their execution order in a task must match

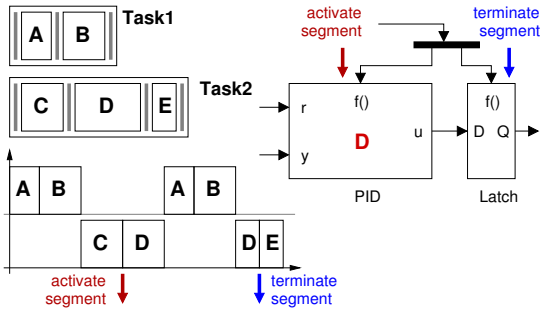


Fig. 2: Segments execution model in T-Res.

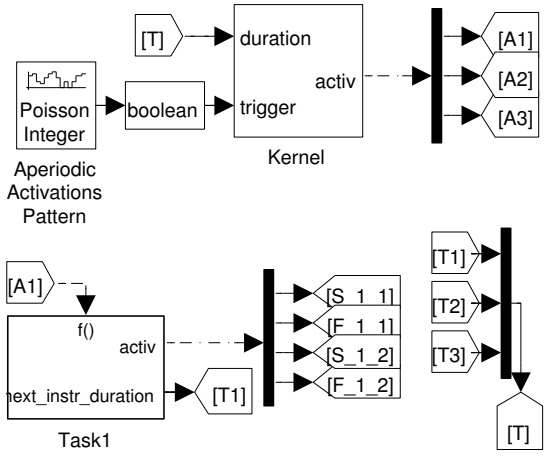


Fig. 3: Interfaces of T-Res `Kernel` and `Task` blocks.

the order of execution of subsystems. The time duration of each segment corresponds to the execution time of the code implementing the subsystem. The start and completion times of the segments correspond to the times in which the corresponding subsystems read or sample their inputs and produce their outputs.

To synchronize in Simulink the completion of the segments with the production of the output values, the activation of the Simulink subsystems is changed from periodic to function activated and a latch barrier is added on all their outputs. Figure 2 shows the activation mechanism: when a segment starts executing, the subsystem is activated; the output signals are latched and enabled when the segment terminates. The signals activating a subsystem (and its input sampling) and its output latch are generated by the task blocks upon the beginning of the execution and the completion of the task segment.

The actual implementation of T-Res relies on two custom blocks, namely `Kernel` and `Task`, implemented as C++ S-Functions. Blocks' input/output interfaces are shown in Figure 3. The block `Task` models one instance of a task that consists of the serialized execution of the segments/subsystems. `Task` is a triggered subsystem, executed on the occurrence of a function call event received on its port `f()`. Its output port `activ` issues activation and termination events to the task segments; port `next_instr_duration` outputs a scalar signal representing the duration of next segment executed by the task. The duration of segments is set by a variable in the Matlab workspace. The computation time of a segment can be fixed or random (e.g., uniform and exponential distributions).

The block `Kernel` models an event-based RT kernel and the scheduler inside it on a single- or multi-core computer node. It is responsible for keeping the scheduling simulation aligned with the system simulation. At each activation, it checks for any aperiodic requests. If there is any, it activates the corresponding aperiodic tasks. Next, it advances the RT scheduler simulator. Two types of events are relevant for the simulation: the *segment completion* and *task completion*. In the first case, `Kernel` reads the input signal on the port `duration` and dynamically creates a new instruction for the corresponding task. In the second case, `Kernel` resets the internal state of the corresponding task. T-Res is open-source and is released under the terms of the 3-Clause BSD License.

A. Finding the maximum subsystem delays using T-Res: the quadcopter example

The quadcopter model of our case study (Figure 1) is evaluated on a path in which it takes off and flies in a circle at constant altitude, while spinning slowly around its Z-axis. The inputs are the speeds of the four rotors; the output is the 12-element state vector with the position, velocity, orientation and orientation rate of the quadcopter. The actual vehicle velocity is assumed to be estimated by an inertial navigation system or GPS receiver (i.e., there is no velocity estimator in the Simulink model).

The control strategy involves multiple nested loops that compute the required thrust and torques so that the quadcopter moves to the desired set-points. Each control loop is a Simulink subsystem executing at its rate. A suffix `_si` is appended at the name of each control subsystem in the figure to identify its code implementation f_i. In the following, for sake of simplicity, we will use the numeric indices of subsystems to refer to the corresponding functions in the SW task implementation.

The periods of the subsystems and functions are: $P_1 = 100$ms, $P_2 = 20$ms, $P_3 = 20$ms, $P_4 = 50$ms, $P_5 = 25$ms and $P_6 = 20$ms.

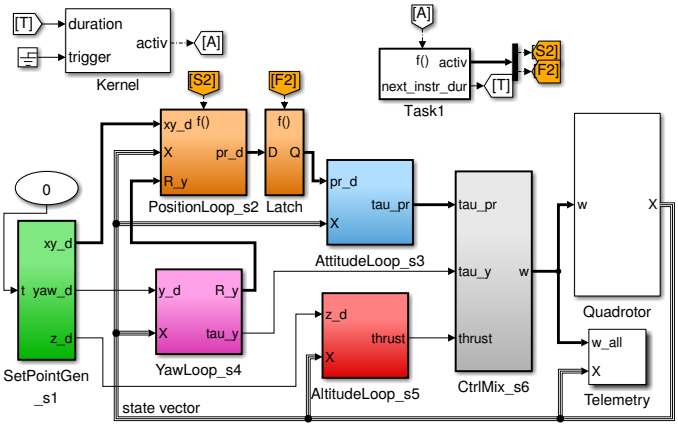


Fig. 4: The T-Res setup to evaluate the maximum acceptable delay for f_2.

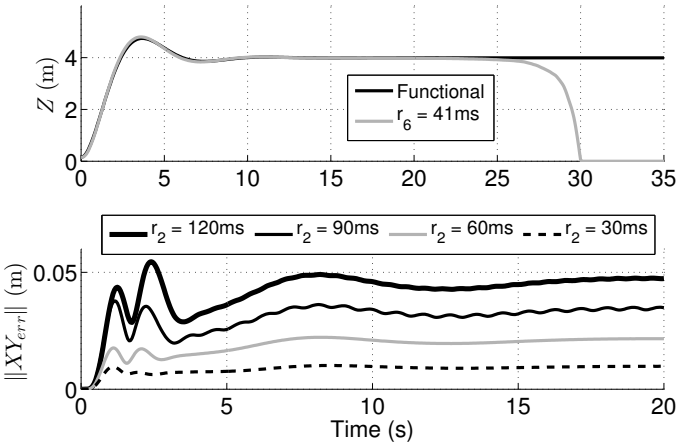


Fig. 5: Values of delays for f_6 (top) and f_2 (bottom) for which the performance is significantly compromised.

A relaxed deadline D_i^{max} is computed for each function f_i by estimating the maximum delay that it can experience in isolation, i.e., assuming an ideal execution of the other functions, before the control performance deteriorates in an unacceptable way. To estimate D_i^{max}, each function f_i is considered as the only time-consuming computation activity in the system and its delay before writing on ports is incremented, until the control becomes unstable, or the deviation from the pure functional control (i.e., the control where *all* functions execute in zero time, including f_i) is so large that the performance is considered unacceptable.

Figure 4 shows the model configuration to compute the relaxed deadline of f_2, by using T-res. A single periodic task (`Task1`), executes f_2 with a variable computation delay/execution time, defined as a variable in the Matlab workspace and incremented across multiple simulations, until an approximation of D_2 is found.

Figure 5 shows the control performance when varying the response time of functions f_6 and f_2 (in isolation), respectively indicated by r_6 and r_2. The top side shows the impact of r_6 on the altitude (Z) control. The control performs well until r_6 is incremented to 41ms, when it suddenly becomes unstable and makes the quadcopter fall down on the ground. The bottom side of Figure 5 shows the effects of increasing r_2 with respect to the XY path following. The graph shows the norm of the difference of the controlled position with respect to the pure functional control, indicated as $\|XY_{err}\|$, versus time. For $r_2 = 30$ms the difference is small at steady-state (black dashed line), and the control performance is acceptable. It becomes larger as r_2 increases, and reaches a significant steady-state value for $r_2 = 90$ms (continuous-thin black line). There are also small peaks in the range 0–5 sec., that indicate a further deviation from the original simulation results in the early phases of the application of the control action. For $r_2 = 120$ms, $\|XY_{err}\| \simeq 5$cm at steady state and peaks are even larger. The control performance is considered unacceptable for $r_2 > 120$ms.

The procedure is repeated for the other functions implementing the flight-control logic. As a result, the relaxed deadlines are: $D_1^{max} = 500$ms, $D_2^{max} = 120$ms, $D_3^{max} = 40$ms, $D_4^{max} = 301$ms, $D_5^{max} = 82$ms and $D_6^{max} = 40$ms.

IV. Optimization model

In order to compute optimal mappings with respect to a given optimization metric, we model the problem as a Mixed Integer Linear Program (MILP).

A MILP formulation is defined by a set of constraints delimiting the set of feasible solutions, and an objective function to optimize. Constraints and objective function are defined in terms of optimization variables (the design parameters to be determined) and parameters (the known values).

For our function allocation and task configuration problem we extend the MILP formulations in [25] [26] [29]. All these papers considered deadlines lower than or equal to periods. We generalize the model to arbitrary deadlines.

Optimization variables

To determine a mapping, a task must be assigned to each function, a priority must be assigned to each task, and an *execution order* must be defined for functions executed in the same task. The task mapping and task priority assignment are defined by a single set of *priority* values assigned to functions. Each priority value is implicitly assigned and identifies a single task. The function priority defines at the same time the task into which it executes and its priority level. Given that the priority assignment defines a total order, we do not assign absolute priority values, but rather a priority order.

Priorities are defined by variables $\pi_{i,j}$, $i, j = 1, \ldots, n$:

$$\pi_{i,j} = \begin{cases} 1 & \text{if } \pi_i > \pi_j \\ 0 & \text{otherwise} \end{cases}$$

A sequence order on functions assigned to the same task (i.e. with the same priority) is defined by variables $\sigma_{i,j}$, $i, j = 1, \ldots, n$:

$$\sigma_{i,j} = \begin{cases} 1 & \text{if } \pi_i = \pi_j \text{ and } f_i \rightarrow f_j \\ 0 & \text{otherwise} \end{cases}$$

The $\pi_{i,j}$ and $\sigma_{i,j}$ assignments must be constrained in such a way that the transitive and antireflexive properties hold for priority and order assignments (omitted here, see [26] for a description).

A. Constraints

A necessary requirement for a mapping is to ensure the schedulability of all the functions, i.e. the following constraints must be satisfied:

$$r_i \leq D_i, i = 1, \ldots, n \tag{4}$$

where r_i denotes the response time of function f_i, $i = 1, \ldots, n$. Response times are computed as described in Section II-A.

Deadlines less than or equal to periods

When the response times are less than the periods, Equation (1) applies. The MILP encoding of (1) is (as in [30])

$$r_i = C_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \sigma_{j,i}\, C_j + \sum_{\substack{j=1 \\ j \neq i}}^{n} \pi_{j,i}\, C_j\, I_{j,i} \tag{5}$$

where integer variable $I_{j,i}$ represents the possible number of interferences of (possibly higher priority) function f_j on f_i. The variable $I_{j,i}$ is defined by the bounds

$$r_i/P_j \leq I_{j,i} < r_i/P_j + 1 \tag{6}$$

Deadlines possibly larger than periods

In this case, Equation (2) is expressed as:

$$\overline{r}_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \overline{r}_i\, \pi_{j,i} U_j + C_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \sigma_{j,i} C_j + \sum_{\substack{j=1 \\ j \neq i}}^{n} \pi_{j,i} C_j\, (1 - U_j) - \gamma_i \tag{7}$$

where:

$$\gamma_i = \sum_{\substack{j=1 \\ j \neq i}}^{n-1} \sum_{\substack{q=j+1 \\ q \neq i}}^{n} \pi_{j,i}\, \pi_{q,i}\, \min(P_j, P_q)\, U_j\, U_q \tag{8}$$

Equations (7) and (8) are not linear (quadratic) due to the products of the optimization variables. To linearize Equation (7), we introduce a real variable $\rho_{j,i}$ that accounts for the product $\overline{r}_i\, \pi_{j,i}$ and is defined using the *big M* formulation that is typically used to encode conditional constraints.

$$\rho_{j,i} = \begin{cases} \overline{r}_i & \text{if } \pi_{j,i} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The value of $\rho_{j,i}$ is determined by the following constraints:

$$\rho_{j,i} \geq 0 \tag{10}$$
$$\rho_{j,i} \leq \overline{r}_i \tag{11}$$
$$\rho_{j,i} \leq M\, \pi_{j,i} \tag{12}$$
$$\rho_{j,i} \geq \overline{r}_i - M\, (1 - \pi_{j,i}) \tag{13}$$

where M is any constant greater than $\overline{r}_i$. A suitable value for M is (from an upper bound on Equation (2)).

$$M = \sum_{j=1}^{n} C_j (2 - U_j) \tag{14}$$

In a similar way, to linearize Equation (8), we introduce the variable $\mu_{j,q,i}$:

$$\mu_{j,q,i} = \begin{cases} 1 & \text{if } \pi_{j,i} = 1 \wedge \pi_{q,i} = 1 \\ 0 & \text{otherwise} \end{cases}$$

and the following constraints:

$$\mu_{j,q,i} \leq \pi_{j,i} \tag{15}$$
$$\mu_{j,q,i} \leq \pi_{q,i} \tag{16}$$
$$\pi_{j,i} + \pi_{q,i} \leq \mu_{j,q,i} + 1 \tag{17}$$

Finally, Equations (7) and (8) are replaced by:

$$\overline{r}_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \rho_{j,i}\, U_j + C_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \sigma_{j,i}\, C_j + \sum_{\substack{j=1 \\ j \neq i}}^{n} \pi_{j,i}\, C_j\, (1 - U_j) - \gamma_i \tag{18}$$

$$\gamma_i = \sum_{\substack{j=1 \\ j \neq i}}^{n-1} \sum_{\substack{q=j+1 \\ q \neq i}}^{n} \mu_{j,q,i}\, \min(P_j, P_q)\, U_j\, U_q \tag{19}$$

In addition, the mapping must be performed in such a way that all functions in a task have the same period. For any pair of functions $\{f_i, f_j\}$ it must be

$\pi_i = \pi_j \Rightarrow P_i = P_j$,

encoded in MILP form by the following constraint:

$$\pi_{i,j} + \pi_{j,i} = 1 \quad \text{for all } i, j = 1, \ldots, n, \quad \text{such that } P_i \neq P_j \tag{20}$$

Finally, the last set of constraints deals with the need of preserving the order of execution of functions. For any pair of functions $\{f_i, f_j\}$ such that $f_i \rightarrow f_j$ in some path, we have:

$$\pi_i \geq \pi_j \vee (\pi_i = \pi_j \wedge \sigma_{i,j} = 1), \tag{21}$$

represented by the following constraints:

For all $i, j = 1, \ldots, n,$ such that $f_i \rightarrow f_j$ in some path:

$$\pi_{j,i} = 0 \tag{22}$$
$$\sigma_{j,i} = 0 \tag{23}$$
$$\sigma_{i,j} = 1 - \pi_{i,j} \tag{24}$$

B. Optimization metrics

We consider three optimization metrics based on path latency, intuitively corresponding to the worst case end-to-end response on a given path. The latency of path p_i is denoted as $\mathcal{L}_i$ and computed as [26]:

$$\mathcal{L}_i = \sum_{f_j \in p_i} P_j + r_j \tag{25}$$

We consider the classical *average latency* and *maximum latency* metrics for minimization.

Metric 1. Average latency (AL): $\frac{1}{q}\sum_{i=1}^{q}\mathcal{L}_i$

Metric 2. Maximum latency (ML): $\max_{i=1,\ldots,q}\mathcal{L}_i$

Another optimization metric that attempts at easing future extensibility, is the maximization of the minimum slack. We considering the fractional slack (relative to the D term), in order to have a normalized objective function.

Metric 3. Maximizing the minimum fractional slack (FS): $\max \min_{i=1,\ldots,q} \frac{D(p_i)-\mathcal{L}_i}{D(p_i)}$, which is equivalent (through simple math) to minimize the maximum relative latency: $\max_{i=1,\ldots,q} \frac{\mathcal{L}_i}{D(p_i)}$.

where $D(p_i)$ is the deadline of path p_i, computed as:

$$D(p_i) = \sum_{f_j \in p_i} P_j + D_j \tag{26}$$

V. Case Study

In the optimization of the task implementation of our quadcopter example, we consider five configurations, with different execution times, each represented in the following table (all times are in ms, the last column shows the total system utilization U_t, ranging from 84% to 99% - overload conditions are not considered):

	C_1	C_2	C_3	C_4	C_5	C_6	U_t
I_{84}	3	4	5	5	4	2	0.84
I_{92}	2	5	5	5	5	2	0.92
I_{94}	3	5	5	8	5	1	0.94
I_{94b}	2	5	5	4	6	2	0.94
I_{99}	2	6	5	4	6	2	0.99
D_i^{min}	100	20	20	50	25	20	
D_i^{max}	500	120	40	301	82	40	

A. Exploration strategies

The first step to the software synthesis is to compute an optimal mapping for each possible configuration, that preserves constraints (4) with $D_i = D_i^{min}$ and (22)–(24) (preservation of execution order). The solver could not compute any feasible mapping for the five configurations. To obtain a feasible mapping, we explored two possible relaxations of the model:

- an execution that violates the order of execution among subsystems, denoted as R_o and,
- an implementation R_d that allows $D_i = D_i^{max}$, $i = 1,\ldots,n$.

In the exploration of the possible solutions, we allow executions that violate the order prescribed by the Simulink semantics (R_o). Order violations may results in data (control samples) loss by overwriting or skipping. The effect is similar to a disturbance that may be tolerated by the control systems.

We therefore solved three optimization problems by combining these two relaxations:

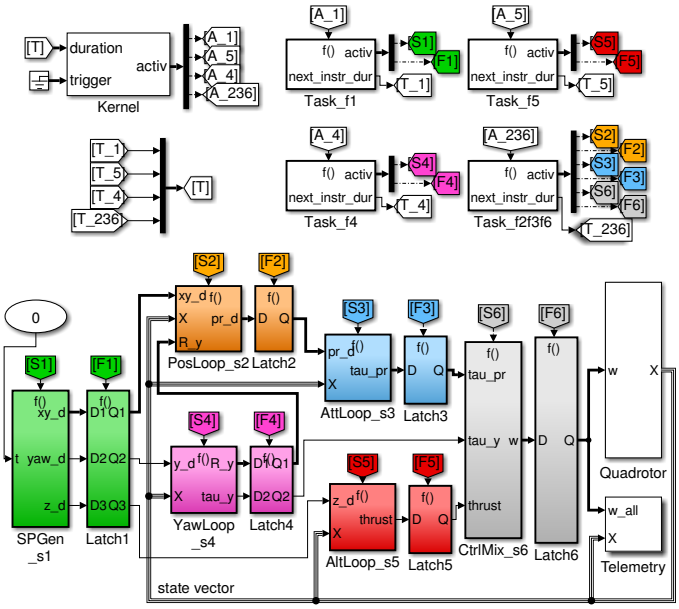


Fig. 6: A task implementation model for the quadcopter example

- Model R_o is obtained by relaxing the execution order preservation constraints (Equations 22, 23, 24);
- Model R_d is obtained by setting $D_i = D_i^{max}$, $i = 1,\ldots,n$;
- Model R_{od} is obtained by relaxing the execution order preservation constraints (Equations 22, 23, 24) and setting $D_i = D_i^{max}$, $i = 1,\ldots,n$.

For each of these three problems we tried the three candidate optimization functions. The nine resulting MILP formulations are solved for each of the five execution time configurations described before. The MILPs are solved with an IBM ILOG Cplex 12.6 solver. The computed mappings and the results are summarized in Table I. In the table, AL indicates the results when maximizing the average latency, ML for the optimization of the maximum latency, and FS for the maximum fractional slack. The notation for the results is as follows. Tasks are listed from higher to lower priority; for each task, the function indexes mapped onto it are shown. For example, [6],[1],[2,3],[5],[4] indicates that the highest priority task executes function f_6, the next task executes function f_1, then another lower priority task executes f_2 and f_3 in sequence, then a task executes f_5 and finally the lowest priority task executes f_4. When multiple optimal solutions are found, they are all listed in the corresponding cell.

The highest utilization configuration is only feasible when $D_i = D_i^{max}$, $i = 1,\ldots,n$. Also, different relaxation methods and different metrics functions can bring to substantially different configurations, even for our relatively simple case study. The task of the performance evaluation stage is to understand which relaxation strategy and which metric function work best.

B. Modeling mappings in Simulink

We use the T-Res co-simulation framework to evaluate the quality of the implementation options (task mappings),

TABLE I: Table of all computed mappings.

		R_o	R_{od}	R_d
I_{84}	AL	[6],[1],[2,3],[5],[4]	[6],[1],[4],[2,3],[5]	[1],[4],[5],[2,3,6]
	ML	[6],[1],[2,3],[5],[4]	[1],[4],[6,2,3],[5]	[1],[4],[5],[2,3,6]
	FS	[6,2,3],[5],[4],[1]	[6,3],[5],[2],[4],[1] [6,3],[5],[1],[2],[4] [6,3],[1],[5],[2],[4] [6,3],[5],[2],[1],[4]	[1],[4],[5],[2,3,6] [1],[5],[4],[2,3,6]
I_{92}	AL	[6,3,2],[5],[4],[1]	[6],[1],[4],[3,2],[5]	[1],[4],[5],[2,3,6]
	ML	[6,2,3],[5],[4],[1] [6,3,2],[5],[4],[1]	[1],[4],[6,2,3],[5]	[1],[4],[5],[2,3,6]
	FS	[6,2],[5],[3],[4],[1] [6,2,3],[5],[4],[1]	[6,3],[5],[2],[1],[4] [6,3],[1],[5],[2],[4] [6,3],[5],[1],[2],[4] [6,3],[5],[2],[4],[1]	[1],[5],[4],[2,3,6] [1],[4],[5],[2,3,6]
I_{94}	AL	[6,3,2],[5],[4],[1]	[6],[1],[4],[3,2],[5]	[1],[4],[5],[2,3,6]
	ML	[6,2,3],[5],[4],[1] [6,3,2],[5],[4],[1]	[6],[1],[4],[2,3],[5] [6],[1],[4],[3,2],[5]	[1],[5],[4],[2,3,6] [1],[4],[5],[2,3,6]
	FS	[6,2],[5],[3],[4],[1] [6,3],[5],[2],[4],[1] [6],[5],[3,2],[4],[1] [6,3,2],[5],[4],[1]	[6,3],[5],[2],[1],[4] [6,3],[1],[5],[2],[4] [6,3],[5],[1],[2],[4]	[1],[4],[5],[2,3,6]
I_{94b}	AL	[6,3,2],[5],[4],[1]	[6],[1],[4],[3,2],[5]	[1],[4],[5],[2,3,6]
	ML	[6,2,3],[5],[4],[1] [6,3,2],[5],[4],[1]	[1],[4],[6,3,2],[5] [1],[4],[6,2,3],[5]	[1],[4],[5],[2,3,6]
	FS	[6,3,2],[5],[4],[1] [6,2,3],[5],[4],[1]	[6,3],[5],[2],[1],[4] [6,3],[5],[1],[2],[4] [6,3],[1],[5],[2],[4]	[1],[5],[4],[2,3,6]
I_{99}	AL	Infeasible	[6],[1],[4],[3,2],[5]	[1],[4],[5],[2,3,6]
	ML		[1],[4],[6,3,2],[5]	[1],[4],[5],[2,3,6] [1],[5],[4],[2,3,6]
	FS		[6,3],[4],[1],[5],[2] [6,3],[5],[1],[4],[2]	[1],[4],[5],[2,3,6]

returned as optimal solutions by the MILP solver, with respect to the control performances. Figure 6 shows the Simulink scheme that models a run-time implementation of the controls with four periodic tasks executed by a priority-based RT kernel (in which functions f_2, f_3, f_6 are realized by one task and all other functions have a dedicated task). `Task_f1` runs every 100ms and reads the set-points (f_1). `Task_f2f3f6` uses the set-points and the current state of the vehicle to perform the position control. Every 20ms, it executes the position loop (f_2), the attitude loop (f_3) and the control mixer (f_6), in sequence. Finally, `Task_f4` and `Task_f5` use the same information to perform yaw (f_4) and altitude control (f_5) with a period of 50ms and 25ms, respectively.

The model in Figure 6 can represent all the mappings in the last column of Table I with a suitable definition of task priorities and functions computation times. The priorities of the tasks and the execution times of the functions are specified in configuration variables in the Matlab workspace and set as parameters of the `Kernel` and `Task` blocks. The top side of Figure 7 shows a snapshot of the Matlab variable that configures `Kernel`. It specifies the task priorities for the

```
% Description of timing properties of task set
task_set_descr = {...
  % type          %iat       %ph  % prio
 'PeriodicTask', 100*0.001, 0.0, 0; ...     % Task_f1
 'PeriodicTask', 25*0.001,  0.0, 10; ...    % Task_f5
 'PeriodicTask', 50*0.001,  0.0, 5; ...     % Task_f4
 'PeriodicTask', 20*0.001,  0.0, 15; ...}; % Task_f2f3f6

% Sequences of pseudo instructions (I94b)
f1_instrs = {'fixed(0.002)'}; % f1
f5_instrs = {'fixed(0.006)'}; % f5
f4_instrs = {'fixed(0.004)'}; % f4
f2f3f6_instrs = {...
 'fixed(0.005)'; ...          % f2
 'fixed(0.005)'; ...          % f3
 'fixed(0.002)'; ... };       % f6
```

Fig. 7: Configuration of `Kernel` and `Task` blocks to model mapping [1], [4], [5], [2,3,6] for I_{94b}.

mapping [1], [4], [5], [2,3,6] (lower numbers indicate higher priorities) and sets the computation time of each function according to instance I_{94b}.

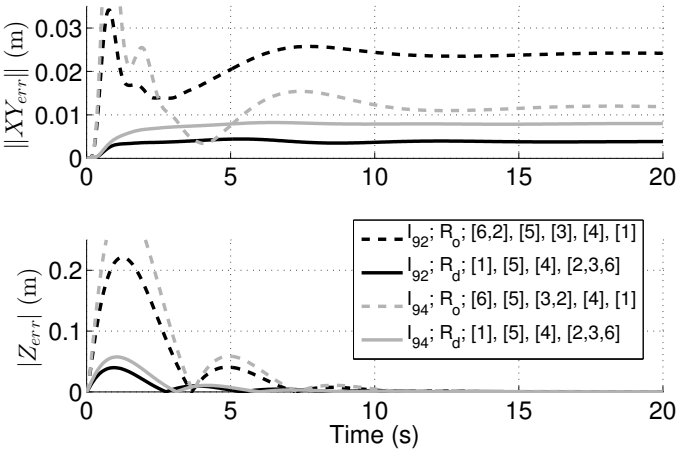
Similar Simulink models and Matlab code configurations of the blocks `Kernel` and `Task` enable the representation of all the mappings in Table I.

C. Performance evaluation

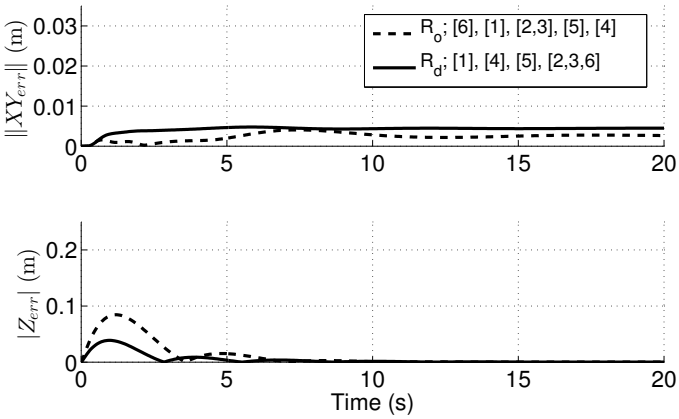
Figures 8–9 show the measure of the absolute difference of the controlled variables values between the task implementations (with execution and scheduling delays) and the pure functional design. On top, the figures show the norm of the difference $\|XY_{err}\|$ in the controlled XY variables (XY-path following). At the bottom, they show the absolute value of the difference of the controlled altitude $|Z_{err}|$. Lower errors indicate better results.

The use of different metrics in R_o and R_d has no significant impact on the control performance. This is probably due to the size of the case study and the restrictions of such models.

Figure 8a shows the results for the mapping obtained by optimizing the *maximum fractional slack* metric. The dashed lines represent the performance when the order of execution is relaxed (R_o), whereas continuous lines correspond to the case of relaxed deadlines (R_d). Only instances I_{92} and I_{94} are shown in the figure, respectively as dark and light lines, since they are representative of the behaviour of all cases R_o and R_d and high processor utilizations ($U \geq 92\%$). R_o generates mappings with worse performances. This is clearly visible for the altitude control. The trajectory control exhibit two kinds of behaviours: either the dashed line is strictly higher than the continuous one ($U = 92\%$), or the two lines nearly converge at steady-state but the dashed one has an initial peak ($U = 94\%$). This is due to f_1 having the lowest priority, which occurs in most R_o mappings (see Table I). In those cases the vehicle is forced to follow a wrong reference trajectory at the beginning of the control application, and this causes a large deviation (peak) from the simulation results obtained with the pure functional design. Function f_1 has the lowest priority in most R_o mappings because of the tight deadlines of the other functions. The low priority of f_1 corresponds to a violation of the execution order. The solutions found with both relaxations



(a) Difference in control accuracy with respect to ideal case: $U \geq 92\%$.



(b) Difference in control accuracy with respect to ideal case: $U = 84\%$

Fig. 8: Difference in control accuracy with respect to ideal case

enabled (R_{od}) (not shown in Figure 8a) yield a performance in between the two R_d and R_o cases.

A lower processor utilization ($U = 84\%$) may increase the solution space, and the model R_o yields to a mapping with a very good performance in terms of trajectory following (dashed line in Figure 8b).

In order to evaluate the performances of the different metrics, we focus on the problem definition R_{od}, which yields a larger solution space. Figure 9 shows the performance of the solutions for case I_{92} (other instances have a similar behaviour). The mapping with the best overall performance (i.e., on both trajectory following and altitude control) is [6,3], [1], [5], [2], [4], obtained by minimizing the *maximum fractional slack*. This mapping produces a good performance for the trajectory following, because the most critical functions f_1 and f_6 have high priorities. It also has a good performance in altitude control, because f_5 is not too much delayed. In case the task of trajectory following is considered to be more critical than altitude control, optimizing the *maximum latency* yields the best performance (dark dashed line in Figure 9). Minimizing the *average latency* seems to yield the least performant mappings (light dashed line). This result is somewhat expected, since this metrics is quite coarse: it does not target

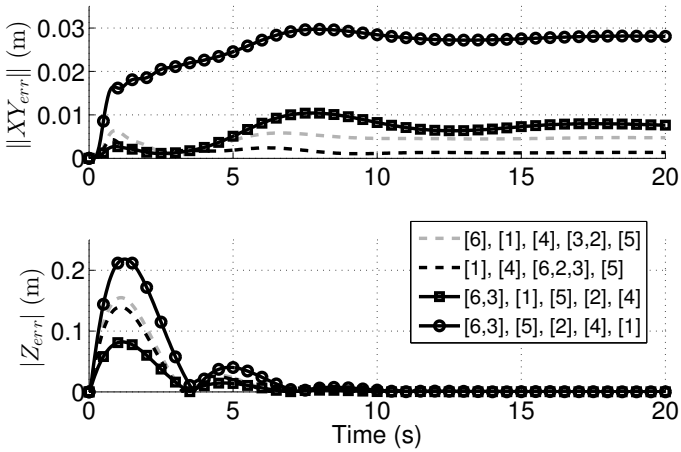


Fig. 9: R_{od} model, comparison of task configurations and optimization metrics ($U = 92\%$)

individual paths and allows some responses to be quite large while others can be very small.

The continuous lines in Figure 9 also show how different mappings that are equivalent from the optimality point of view for a given objective function (the *maximum fractional slack* in this case) may exhibit different control performances. This enforces the claim that it is important to co-design the system and not simply design the control first and then synthetize its SW implementation, by merely dealing with the real-time aspects and concerns of the design.

VI. Conclusions and Future Work

This paper presented a simulation-driven optimization process for the implementation of real-time control software. The experiments conducted on a quadcopter case study have clearly shown that the consideration of the control performance for real-time software design leads to better results. The presented experiments involved several tools with many manual operations. In order to fluidize the design and optimization process, we aim to setup a toolchain integrating the design and simulation tools with optimization engines. This work based on model-driven solutions is ongoing. The resulting automated toolchain will be used to perform a large number of experiments. We intend to use the optimization results of these experiments to identify efficient design patterns for real-time control applications. Future investigation will include the study of allocation policies and different scheduling options in multi-core architectures.

References

[1] M. Joseph and P. K. Pandya, "Finding response times in a real-time system", in *The Computer Journal* 29 (5), 390–395, 1986.

[2] E. Bini, A. Parri, G. Dossena, "A quadratic time response time upper-bound with a tightness property". Proceedings of the IEEE Real Time Systems Symposium, 2015.

[3] D. Seto, J. Lehoczky, L. Sha, and K. Shin, "On task schedulability in real-time control systems," in *Real-Time Systems Symposium, 1996., 17th IEEE*, dec 1996, pp. 13 –21.

[4] K. J. Astr om and B. Wittenmark, "Theory and design," in *Computer-Controlled Systems*. Prentice Hall, 2007.

[5] E. Bini and A. Cervin, "Delay-aware period assignment in control systems," in *Real-Time Systems Symposium, 2008*, 30 2008-dec. 3 2008, pp. 291 –300.

[6] S. Samii, A. Cervin, P. Eles, and Z. Peng, "Integrated scheduling and synthesis of control applications on distributed embedded systems," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 57 –62.

[7] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty, "Time-triggered implementations of mixed-criticality automotive software," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 1227 –1232.

[8] Simulink, "http://www.mathworks.com/products/simulink/."

[9] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 2, 2002, pp. 1211–1217 vol.2.

[10] S. Samii, P. Eles, Z. Peng, and A. Cervin, "Design optimization and synthesis of flexray parameters for embedded control applications," in *Electronic Design, Test and Application (DELTA), 2011 Sixth IEEE International Symposium on*, 2011, pp. 66–71.

[11] S. Samii, P. Eles, Z. Peng, P. Tabuada, and A. Cervin, "Dynamic scheduling and control-quality optimization of self-triggered control applications," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, 30 2010-dec. 3 2010, pp. 95 –104.

[12] A. Aminifar, P. Eles, Z. Peng., and A. Cervin, "Control-quality driven design of cyber-physical systems with robustness guarantees," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013.

[13] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, Jan 2003.

[14] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, june 2007, pp. 278 –283.

[15] D. Goswami, R. Schneider, and S. Chakraborty, "Co-design of cyber-physical systems via controllers with flexible delay constraints," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, jan. 2011, pp. 225 –230.

[16] L. Sha, X. Liu, M. Caccamo, and G. Buttazzo, "Online control optimization using load driven scheduling," in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, vol. 5, 2000, pp. 4877 –4882 vol.5.

[17] X. Liu, Q. Wang, S. Gopalakrishnan, W. He, L. Sha, H. Ding, and K. Lee, "Ortega: An efficient and flexible online fault tolerance architecture for real-time control systems," *Industrial Informatics, IEEE Transactions on*, vol. 4, no. 4, pp. 213 –224, nov. 2008.

[18] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, no. 3, pp. 239–272, 2007.

[19] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," in *Optimization and Engineering*, april 2007, pp. 67 –127.

[20] M. Neukirchner, S. Stein, and R. Ernst, "Smff: System models for free," in *2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2011)*, 2011.

[21] F Cremona, M Morelli, M Di Natale, "TRES: A Modular Representation of Schedulers, Tasks, and Messages to Control Simulations in Simulink" in *Proc. of the ACM SAC Conference*, 2015, Salamanca, Spain.

[22] P.I. Corke, "Robotics, Vision and Control: Fundamental Algorithms in Matlab." *Springer Publisher*, 2011.

[23] M. Di Natale, V. Pappalardo, "Buffer optimization in multitask implementations of simulink models" in *ACM Transactions on Embedded Computing Systems (TECS)*, 2008, 7 (3), 23.

[24] J.P. Lehoczky. "Fixed priority scheduling of periodic task sets with arbitrary deadline." In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, 1990. Lake Buena Vista, FL USA, 201209.

[25] A. Metzner, and C. Herde, "Rtsat an optimal and efficient approach to the task allocation problem in distributed architectures". In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, 2006. Washington, DC, USA, 147158.

[26] Q. Zhu, H. Zeng, W. Zheng, M. Di Natale, A. Sangiovanni-Vincentelli "Optimization of task allocation and priority assignment in hard real-time distributed systems", in *ACM Transactions on Embedded Computing Systems*, 2012, (TECS) 11 (4), 85.

[27] Q. Zhu, Y. Yang, M. Di Natale, E. Scholte, A. Sangiovanni-Vincentelli "Optimizing the software architecture for extensibility in hard real-time distributed systems", in *IEEE Transactions on Industrial Informatics,* 6 (4), 621-636

[28] G. Wang, M. Di Natale, A. Sangiovanni-Vincentelli "Improving the size of communication buffers in synchronous models with time constraints", in *IEEE Transactions on Industrial Informatics,* 5 (3), 229-240.

[29] A. Mehiaoui, E. Wozniak, S. Tucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, J.-P. Babau, L. Lemarchand, S. Gerard, "A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems", in *Proceedings of the ACM/IEEE LCTES Conference*, 2013.

[30] H Zeng, M Di Natale, "An efficient formulation of the real-time feasibility region for design optimization", in *IEEE Transactions on Computer*, 2013, 62 (4), 644-661.

[31] Cervin, A., Henriksson, D., Lincoln, B., Eker, J., Arzen, K.E. "How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime", in *IEEE control systems* 23(3), 1630 (June 2003).