Selecting the Transition Speeds of Engine Control Tasks to Optimize the Performance

ALESSANDRO BIONDI, MARCO DI NATALE, GIORGIO C. BUTTAZZO, and PAOLO PAZZAGLIA, Scuola Superiore Sant'Anna

Engine control applications include functions that need to be executed at specific rotation angles of the crankshaft. The tasks performing these functions are activated at variable rates and are programmed to be adaptive with respect to the rotation speed of the engine to avoid overloading the CPU. Simplified control implementations are used at high speeds; for example, reducing the number of fuel injections or the complexity of the computations. Such different control implementations define execution modes with different execution times for different ranges of the rotation speed. The selection of the switching speeds for the operating modes of such tasks is an optimization problem, consisting in determining the optimal transition speeds that maximize the engine performance while guaranteeing schedulability.

This article presents three methods for tackling such an optimization problem under a set of assumptions about the performance metrics: two heuristics and a branch and bound method that guarantees finding the optimal solution within a given speed granularity. In addition, a simple method to compute a performance upper bound is presented. The approach and the hypothesis are validated using a Simulink model of the engine and the computational tasks, considering the engine efficiency and the production of pollutants (NO_2) as metrics of interest. Simulation experiments show that the performance of proposed heuristics is quite close to that of the upper bound and the optimum within a finite granularity.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; **Real-time systems**;

Additional Key Words and Pharses: Engine control, design optimization, real-time systems, control performance

ACM Reference format:
Alessandro Biondi, Marco Di Natale, Giorgio C. Buttazzo, and Paolo Pazzaglia. 2018. Selecting the Transition Speeds of Engine Control Tasks to Optimize the Performance. *ACM Trans. Cyber-Phys. Syst.* 2, 1, Article 1 (January 2018), 26 pages.
https://doi.org/10.1145/3127022

1 INTRODUCTION

Engine control is one the most challenging examples of a Cyber-Physical System (CPS), where the software that controls injection and combustion must be designed to take several physical characteristics of the engine into account. From a timing perspective, engine control software requires the execution of different types of computations, some cyclically activated at fixed intervals (*periodic tasks*) ranging from a few milliseconds up to 100ms, and some triggered at predefined rotation

Authors' addresses: A. Biondi, M. D. Natale, G. C. Buttazzo, and P. Pazzaglia, Via G. Moruzzi 1, Pisa, Italy; emails: {alessandro.biondi, marco, giorgio, p.pazzaglia}@sssup.it.

https://doi.org/10.1145/3127022

angles of the crankshaft (*angular tasks*) (Guzzella and Onder 2010). Such angular tasks generate a dynamic workload strictly dependent on the actual engine speed.

To avoid overloading the processor at high engine speeds, angular tasks are typically implemented as a set of operational modes, each characterized by a different computational demand and acting on a different speed interval (Buttle 2012). Since angular tasks adapt their functionality with the speed and are activated at non-constant rate, they are also referred to as *adaptive variable rate* (AVR). The schedulability analysis of AVR tasks has received significant attention from the research community (a detailed review of the literature is available in Section 9).

The implementation of an AVR task requires determining the precise engine speeds at which mode changes should occur. This problem has been addressed under Earliest Deadline First (EDF) scheduling by Buttazzo et al. (2014), who proposed a method for identifying the highest switching speeds that bound the utilization of an AVR task to a desired value.

In reality, the selection of the transition speeds is not driven by schedulability constraints alone, but has to optimize a set of performance indexes, related to power, fuel consumption, and emissions (among others). The control implementations are designed to achieve the best possible combination of all the performance indexes within a given computation complexity and for a given set of engine speeds. This process requires the tuning of a significant number of configuration parameters and is performed at the test bench, where each implementation is tested for different speeds recording the performance parameters of interest.

Intuitively, the most sophisticated control implementations have the best performance but they require a higher computational demand that cannot be afforded (without incurring deadline misses) when they are executed more frequently (i.e., at high engine speeds). Conversely, simpler control implementations have lower computational requirements and tend to work better at higher rotational speeds, where the engine is more stable. The resulting behavior of engine-control tasks is a mode change among a set of different control implementations, each one executed in a given interval of engine speeds.

Article contributions. The main contribution of this article is to formulate a design optimization problem to find the switching speeds that maximize the system performance while guaranteeing the schedulability of the task set. We assume the knowledge of the performance function associated to each control implementation (for instance, they can be derived by fitting the test bench performance data to a family of relatively simple analytical functions). To the best of our knowledge, this is the first work that integrates performance metrics together with real-time constraints on the computational activities to address the design optimization of engine control systems. This article presents three methods for tackling such an optimization problem assuming a given performance metric and a set of constraints. Two heuristic approaches are first proposed to reduce the computational complexity and a performance upper bound is computed. Then, a branch and bound method is presented to find the optimal solution within a given speed granularity. Simulation results show that the performance achieved by the heuristics is quite close to the performance upper bound and the optimum with respect to a given granularity. The performance functions considered in the proposed article have been also validated in a simulation environment that includes a diesel engine model.

Article structure. The remainder of the article is structured as follows. Section 2 presents the model and the notation used throughout the article. Section 3 formally states the problem considered in the article. Section 4 presents a technique to reduce the design space and compute a bound on the maximum performance of the engine. Section 5 presents the two heuristic approaches. Section 6 describes the branch and bound algorithm. Section 7 illustrates a set of experimental results aimed at comparing the proposed approaches. Section 8 reports on the experimental

validation of the performance model considered in this article. Section 9 discusses the related work. Finally, Section 10 states our conclusions and future work.

A preliminary version of this article is available in Biondi et al. (2016), which has been extended by including (i) the validation of the performance model in Section 8; (ii) the pseudo-code (with the corresponding description) for the branch and bound algorithm presented in Section 6; (iii) the proof for the performance upper bound discussed in Section 4; and (iv) additional experimental results in Section 7. Furthermore, the entire article has been polished and, when needed, restructured. Clarifications of some concepts with a more verbose discussion have also been added.

2 SYSTEM MODEL AND BACKGROUND

This article considers applications consisting of a set n real-time preemptive tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task can be a regular *periodic* task, or an AVR task, activated at specific crankshaft rotation angles. Whenever needed, an AVR task may also be denoted as τ_i^*. The rotation source triggering the AVR tasks is characterized by the following state variables:

- θ the current rotation angle of the crankshaft;
- ω the current angular speed of the crankshaft;
- α the current angular acceleration of the crankshaft.

The rotation speed ω is assumed to be limited within a range $[\omega^{min}, \omega^{max}]$ and the acceleration α is assumed to be limited within a range $[\alpha^-, \alpha^+]$.

Both periodic and AVR tasks are characterized by a *Worst-Case Execution Time* (WCET) C_i, an interarrival time (or period) T_i, and a relative deadline D_i. However, while for regular periodic tasks such parameters are fixed, for angular tasks they depend on the engine rotation speed ω. An AVR task τ_i^* is characterized by an *angular period* Θ_i and an *angular phase* Φ_i, so that it is activated at the following angles: $\theta_i = \Phi_i + k\Theta_i$, for $k = 0, 1, 2, \ldots$.

This means that the interarrival of an AVR task is inversely proportional to the engine speed ω and, in steady-state conditions, can be expressed as

$$T_i(\omega) = \frac{\Theta_i}{\omega}. \tag{1}$$

An angular task τ_i^* is also characterized by a relative *angular deadline* Δ_i expressed as a fraction δ_i of the angular period ($\delta_i \in [0, 1]$). In the following, $\Delta_i = \delta_i\Theta_i$ represents the relative angular deadline. All angular phases Φ_i are relative to a reference position called *Top Dead Center* (TDC) corresponding to the crankshaft angle for which at least one piston is at the highest position in its cylinder. Without loss of generality, the TDC position is assumed to be at $\theta = 0$.

As explained in the Introduction, an AVR task τ_i^* is typically implemented as a set $\mathcal{M}_i$ of M_i execution modes. Each mode m has a different WCET C_i^m and operates in a predetermined range of engine speeds $(\omega_i^{m+1}, \omega_i^m]$, where $\omega_i^{M_i+1} = \omega^{min}$ and $\omega_i^1 = \omega^{max}$. Hence, the set of modes of task τ_i^* can be expressed as

$$\mathcal{M}_i = \{(C_i^m, \omega_i^m), m = 1, 2, \ldots, M_i\}.$$

The vector of the set of switching speeds is denoted as $\vec{\omega}_i$, while the vector of WCETs is denoted as $\vec{C}_i$.

We assume that the worst-case execution time of an AVR task τ_i^* can be expressed as a non-increasing step function $C_i(\omega)$ of the instantaneous speed ω at its release, that is,

$$C_i(\omega) \in \left\{C_i^1, \ldots, C_i^{M_i}\right\}. \tag{2}$$

An example of a $C(\omega)$ function is illustrated in Figure 1.

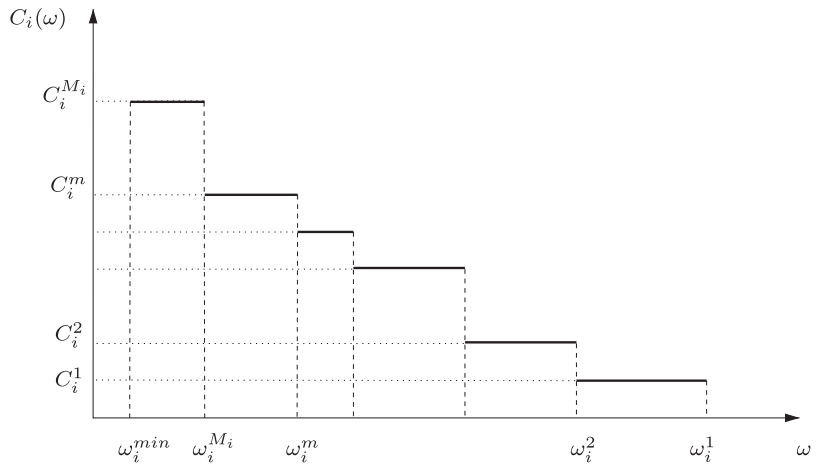

Fig. 1. WCET of an AVR task as a function of the engine speed at the job activation.

For the sake of readability, when a single AVR task is addressed, the task index is omitted. To support the presentation of the following results, it is convenient to define the steady-state utilization of the j^{th} execution mode as a function of an arbitrary switching speed w, that is,

$$U^j(\omega) = \frac{C^j}{\Theta}\omega. \tag{3}$$

Moreover, $U^j = U^j(\omega^j)$ denotes the steady-state utilization of the j-th execution mode at its switching speed ω^j.

2.1 Schedulability Analysis of AVR Tasks

In this article, the schedulability analysis of mixed task sets consisting of periodic tasks and AVR tasks is performed by adopting the technique presented in Biondi et al. (2015).

Considering the speed domain as a continuum, a schedulability test for task sets including AVR tasks must take into account all possible speed evolutions of the rotation source to cope with potential worst-case situations resulting in deadline misses. The adaptive behavior of AVR tasks, which causes mode changes as a function of the engine speed, further complicates the identification of worst-case response times. This prevents treating such tasks as classical periodic/sporadic tasks to apply standard real-time analysis techniques.

For this reason, a novel analysis approach has been proposed in Biondi et al. (2015) to explicitly take into account physical constraints (i.e., the rotation of the crankshaft) when dealing with the mode change issue. Similarly to other standard techniques, such an analysis is based on the computation of the interference caused by higher-priority tasks, which is then used to compute the tasks response times. By restricting to a finite set of dominant speeds (Biondi et al. 2014), it is possible to limit the number of speed evolution patterns that have to be examined for deriving the worst-case interference. The response time computation is then approached as a search problem in the speed domain: the notion of dominant speeds is first used to reduce the number of critical instants (i.e., the situation in which all tasks are released simultaneously) and then the search space is reduced to dominant speeds only.

3 PROBLEM DEFINITION

The analysis presented in this article is restricted to a representative class of automotive applications consisting of a single AVR task and a set of periodic tasks, scheduled by fixed priority on a single core. Since there is a single AVR task, we drop its index in the definition of its parameters.

The addressed problem is first stated using a general formulation and then restricted to a number of specific cases under a set of assumptions. The general formulation consists of the definition of the input parameters, the optimization variables, the set of constraint functions, and the performance function to be optimized.

Input Parameters. We consider an AVR task consisting of Q implementations Λ^j ($j = 1, \ldots, Q$) of the same functionality at different complexity. Note that we distinguish between implementations and mode, since (after the design process) one or more implementations can be merged into a single mode, to be executed in a range of speeds (to be determined) and characterized by a known WCET C^j. The implementations are indexed such that their execution times are strictly increasing with the index j—that is, $\forall j = 1, \ldots, Q-1, C^j < C^{j+1}$.

Optimization Variables. The objective of the proposed optimization algorithms is to find the set of switching speeds between modes of the AVR task that *maximizes* the engine performance while guaranteeing the task set schedulability. The switching speeds $\omega^j \in [\omega^{min}, \omega^{max}]$, with $j = 1, \ldots, Q-1$, are hence the main *variables* of the considered optimization problem. The schedulability condition yields a fixed-priority assignment for the AVR task and the periodic tasks. Note that, in the presence of AVR tasks, a rate-monotonic priority assignment cannot be enforced, due to the large variation of interarrival times of an AVR task (Biondi et al. 2015).

Constraints. The main constraint considered in the optimization procedure is the schedulability of the task set, that is, the condition under which *all* the tasks in the system meet their deadlines. More specifically, the following notion of schedulability is considered in this article: a task set is said to be *schedulable* if there exists at least a fixed-priority assignment such that all the tasks meet their deadlines, otherwise the task set is said to be *unschedulable*. To verify the schedulability of a given instance of task system, we rely on the analysis presented in Biondi et al. (2015) (summarized in Section 2.1).

Performance Metrics. The effectiveness of a design solution is evaluated by assigning each implementation a performance function $f_j(\omega)$ of the engine speed ω, which is monotonically increasing with the WCET C^j—that is, $\forall \omega, \tilde{f}_j(\omega) > f_{j'}(\omega) \Leftrightarrow C^j > C^{j'}$. The rationale behind such a restriction is that a more complex control implementation, which has a higher computational demand (and hence a higher WCET), makes sense only if it improves the performance.

The overall system performance is defined as the integral of the resulting performance function over the entire speed range. Since each execution mode is active in a particular speed range, the overall performance is expressed as the sum of the contributions given by each mode j, whose performance function $f_j(\omega)$ is integrated in its operating range—hence obtaining

$$\mathbb{P}(\omega^1, \ldots, \omega^Q) = \sum_{j=1}^{Q} \int_{\omega^{j+1}}^{\omega^j} f_j(\omega)\, d\omega, \tag{4}$$

where j spans over all the modes $j = 1, \ldots, Q$, and, by definition, $\omega^1 = \omega^{max}$ and $\omega^{Q+1} = \omega^{min}$. In this article, two families of performance functions are considered. In the first one, each implementation brings a contribution to the performance that does not depend on the rotation speed at which it is applied. Each Λ^j has an associated performance constant k^j and the overall system

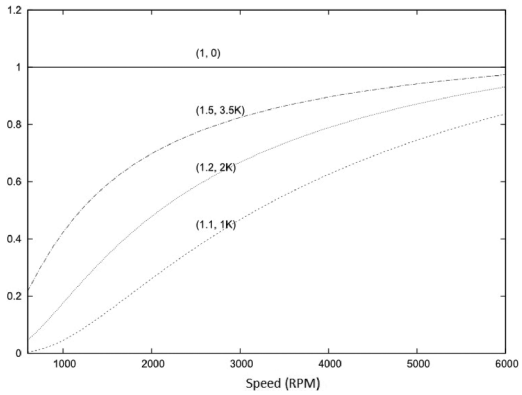

Fig. 2. A set of exponential performance functions for selected values of $k^{j,1}, k^{j,2}$.

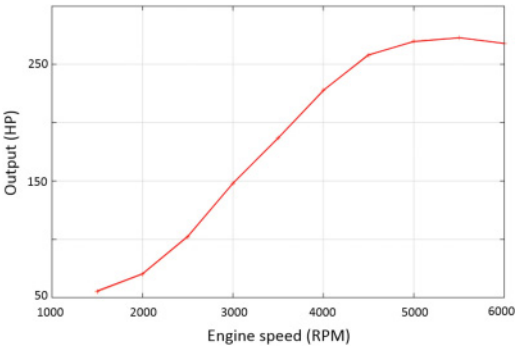

Fig. 3. The power output curve of an Evinrude E-TEC-250 engine (from continuouswave.com).

performance is

$$\mathbb{P}(\omega^1, \ldots, \omega^Q) = \sum_{j=1}^{Q} k^j(\omega^j - \omega^{j+1}). \tag{5}$$

In the second family, functions represent control implementations in which the performance depends on the implemented control law and the rotation speed at which it is applied. The selected function (for each mode) is

$$f_j(\omega) = k^{j,1} e^{-\frac{k^{j,2}}{\omega}} \tag{6}$$

and the system performance is computed as

$$\mathbb{P}(\omega^1, \ldots, \omega^Q) = \sum_{j=1}^{Q} \int_{\omega^{j+1}}^{\omega^j} k^{j,1} e^{-\frac{k^{j,2}}{\omega}} \, d\omega. \tag{7}$$

The rationale for selecting an exponential function of this type is the following. The most sophisticated control implementation ($j = Q$, with possibly multiple injections during the cycle) has the best performance, which is kept unchanged across the speed range (i.e., $f_Q(\omega) = 1$, with $k^{Q,2} = 0$), and is considered as a baseline for the other control implementations. Simplified control implementations tend to work better at higher rotational speeds, where the engine is more stable, but become less effective at some cutoff speed, represented by the parameter $k^{i,2}$. The parameter $k^{i,1}$ is used as an additional degree of freedom to improve fitting the performance curve to the actual experimental data. Figure 2 shows the typical shape of our exponential performance functions for different values of $k^{j,1}, k^{j,2}$, and Figure 3 shows technical data (from the Web) expressing the engine output power as a function of the engine speed, in support of a possible fit with a family of exponential functions as in Equation (6).

Formally, the first type of function is a special case of the exponential one where $k^{j,2} = 0$ for all j. However, in our experiments they are handled in a different way, because the optimization algorithms leverage the constant gradient of the constant functions to avoid recomputing it at every step, thus speeding up the computation.

Although these two classes of functions are used in the experiments, the proposed method is not limited to their consideration. The only requirement is that the used function is monotonic with the mode index (and hence with the mode WCET) for all ω and it is integrable in the speed domain ω.

Table 1. Parameters of the Task Set Used in the Example

	Task1	Task2	Task3	Task4
C	1,000	6,500	10,000	10,000
T=D	5,000	20,000	50,000	100,000

Table 2. Computation Times of the AVR Task in the Example

	Λ^1	Λ^2	Λ^3	Λ^4	Λ^5	Λ^6
C^j	$s\cdot$ 150	$s\cdot$ 278	$s\cdot$ 344	$s\cdot$ 425	$s\cdot$ 576	$s\cdot$ 966
k^j	2	3	4	5	7	10

The integral in Equation (7) cannot be computed analytically, and thus is computed numerically through the *exponential integral* function $Ei(x)$, so obtaining

$$\mathbb{P}(\omega^1, \ldots, \omega^Q) = \sum_{j=1}^{Q} Y_j(\omega^j) - Y_j(\omega^{j+1}), \tag{8}$$

where

$$Y_j(\omega) = k^{j,1}\left(k^{j,2} \cdot Ei\left(\frac{-k^{j,2}}{\omega}\right) + \omega \cdot e^{\frac{-k^{j,2}}{\omega}}\right). \tag{9}$$

For notational convenience, we introduce a shortcut $p^j(\vec{\omega})$ for the partial derivatives of the performance function with respect to the switching speeds, defined as

$$p^j(\vec{\omega}) = \frac{\partial \mathbb{P}(\vec{\omega})}{\partial \omega^j}. \tag{10}$$

Design Objective. The design goal is to compute the set of switching speeds $\{\omega^j, j = 1, \ldots, Q\}$ and the multidimensional optimization problem can be stated as follows:

Definition 3.1 (Optimization problem).

$$\max \mathbb{P}(\omega^1, \ldots, \omega^Q)$$
$$\omega^1 > \omega^2 > \ldots > \omega^Q$$

such that the system is schedulable.

3.1 Running Example

To illustrate the outcome of each algorithm presented in the article, we use a running example (using a configuration from the EU INTERESTED project (INTERESTED)) consisting of four periodic tasks with overall utilization $U = 0.825$, and an AVR task with six modes. The periods and the computation times of the periodic tasks are reported in Table 1, whereas the computation times of the six modes of the AVR task are reported in Table 2. To better explore the input space, the computation times of the AVR task are expressed as a function of a scaling factor s. In the following, two subcases are generated, using $s = 6.0$ and $s = 8.0$ (all times are expressed in microseconds).

The running example has only the purpose of explaining the typical outcome of the proposed algorithms and is only considered for the first set of performance functions (constant over the entire speed range). The experiments in the Experimental section (Section 7) have been conducted

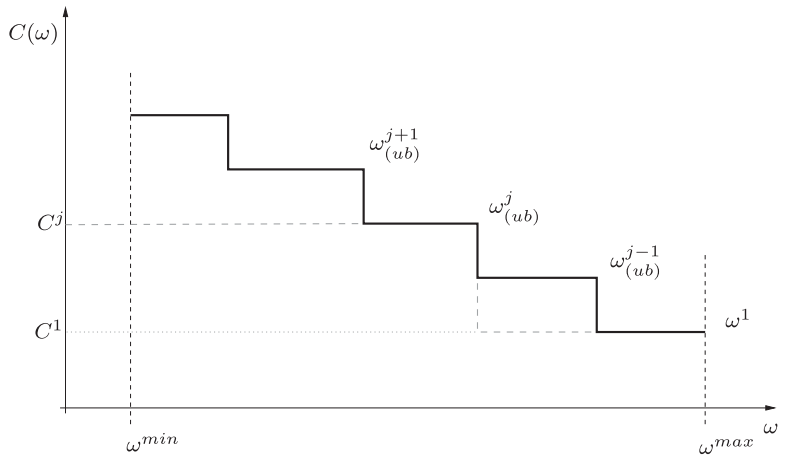

Fig. 4. Computing the switching speed upper bounds.

also for exponential performance functions. In our running example, the coefficients k^j of the performance functions have been set as $k^j \in \{2, 3, 4, 5, 7, 10\}$.

4 REDUCING THE DESIGN SPACE

By exploring the design space, it has been experimentally observed that the schedulability constraint determines a non-convex region in the space of the switching speeds ω^j. This result was not surprising considering that the schedulability region for classical periodic tasks in the space of interarrival times is known to be non-convex (Bini et al. 2008) and, to the best of our knowledge, there is no known convex upper or lower bound of good quality for checking the schedulability of AVR tasks under fixed-priority scheduling.

Although an analytical description of the feasibility region cannot be derived in a closed form, we provide a numerical method for computing (with an arbitrary accuracy) an upper-bound $\omega^j_{(ub)}$ of the maximum speed at which each control implementation can be executed, thus limiting the design space. The upper bound is such that no feasible solution for the optimization problem exists for speeds $\omega^j > w^j_{(ub)}$, with respect to the fixed accuracy.

The proposed algorithm probes the schedulability region in the most favorable condition for each possible control implementation j; that is, when (i) no more complex control implementations are active and (ii) the AVR task performs a direct mode change to the simplest control implementation with WCET C^1, for any $\omega > \omega^j$. An example of such upper bound is shown in Figure 4, in which the mode configuration for determining $\omega^j_{(ub)}$ is shown as a dashed line. Under this condition, the algorithm computes the maximum speed $\omega^j_{(ub)}$ at which each control implementation Λ^j can be executed. The (typically unfeasible) modes configuration for the AVR task, obtained by considering all the upper bounds on the switching speeds, is shown as a thick line in Figure 4.

The approach is guaranteed to be correct because of the sustainability property of uniprocessor fixed-priority scheduling (Burns and Baruah 2008), which states that schedulability never improves when computation times are increased.

The algorithm to compute the switching speed upper bounds is shown in Figure 5. A first step of pre-processing (omitted in the figure for simplicity) can be used to determine whether there are modes that can be safely avoided. This is done by finding (i) the implementation Λ^j with the

1: **procedure** ComputeUBs($\vec{C}$)
2: $w^1_{(ub)} = \omega^{max}$;
3: **for** $j = 2$ **to** $j = Q$ **do**
4: $\mathcal{M} \leftarrow \{(C^1, \omega^{max}), (C^j, \omega^{min})\}$;
5: $\omega^j_{(ub)} =$ MaxBinSearch$(j, \mathcal{M})$;
6: **end for**
7: **return** $\{\omega^j_{(ub)}, j = 1, \ldots, Q\}$;
8: **end procedure**

Fig. 5. Procedure for computing the upper-bounds for the switching speeds.

largest C^j that can safely be executed in ω^{min} and discarding those with *higher* index; and (ii) the implementation $\Lambda^{j'}$ with the largest $C^{j'}$ that is feasible in ω^{max} and discarding those with *lower* index.

For simplicity, in the following, we assume that the only control implementation feasible at ω^{max} is the first one ($j = 1$) and that the control implementation with the largest WCET C^Q is feasible at ω^{min} (that is, the number of modes are equal to the number of available implementations $Q = M$).

The first control implementation has maximum speed $w^1_{(ub)} = \omega^{max}$, since it is not possible to exceed the allowed speed range (see line 2 in Figure 5). The maximum speeds are then determined for each mode $j > 1$ by a simple binary search (line 5) on the schedulability condition, by seeking a speed $\omega^j_{(ub)}$ such that mode C^j is executed in $[\omega^{min}, \omega^j_{(ub)}]$ and mode C^1 is active in $(\omega^j_{(ub)}, \omega^{max}]$.

4.1 Performance Upper Bound

The computation of the upper bound on the switching speeds also leads to an upper-bound $\mathbb{P}_{(ub)}$ on the maximum system performance, which is expressed by the following theorem.

Theorem 4.1. *Any valid solution for the optimization problem expressed in Definition 3.1 has an overall performance no greater than*

$$\mathbb{P}_{(ub)} = \mathbb{P}\left(w^1_{(ub)}, \ldots, w^Q_{(ub)}\right). \tag{11}$$

Proof. Let $\vec{\omega}_{(opt)}$ be the vector of the switching speeds that lead to the optimal solution of the optimization problem and let $\mathbb{P}_{(opt)} = \mathbb{P}(w^1_{(opt)}, \ldots, w^Q_{(opt)})$ be the maximum performance. By Equation (4) we get

$$\mathbb{P}_{(opt)} = \int_{\omega^{min}}^{\omega^Q_{(opt)}} f_Q(\omega)\, d\omega + \int_{\omega^Q_{(opt)}}^{\omega^{Q-1}_{(opt)}} f_{Q-1}(\omega)\, d\omega + \cdots + \int_{\omega^2_{(opt)}}^{\omega^1_{(opt)}} f_1(\omega)\, d\omega.$$

Since the condition $\forall j,\ \omega^j \le w^j_{(ub)}$ holds for any valid solution, then the condition $\forall j,\ \omega^j_{(opt)} \le w^j_{(ub)}$ also holds, which allows rewriting $\mathbb{P}_{(opt)}$ as

$$\mathbb{P}_{(opt)} = \int_{\omega^{min}}^{\omega^Q_{(opt)}} f_Q(\omega)\, d\omega + \int_{\omega^Q_{(opt)}}^{\omega^Q_{(ub)}} f_{Q-1}(\omega)\, d\omega + \int_{\omega^Q_{(ub)}}^{\omega^{Q-1}_{(opt)}} f_{Q-1}(\omega)\, d\omega + \int_{\omega^{Q-1}_{(opt)}}^{\omega^{Q-1}_{(ub)}} f_{Q-2}(\omega)\, d\omega + \cdots .$$

By recalling the assumption on the monotonicity of the performance functions with the mode index—that is, $\forall \omega, f_j(\omega) > f_{j-1}(\omega)$—we get

$$\mathbb{P}_{(opt)} < \int_{\omega^{min}}^{\omega^Q_{(opt)}} f_Q(\omega)\, d\omega + \int_{\omega^Q_{(opt)}}^{\omega^Q_{(ub)}} f_Q(\omega)\, d\omega + \int_{\omega^Q_{(ub)}}^{\omega^{Q-1}_{(opt)}} f_{Q-1}(\omega)\, d\omega + \int_{\omega^{Q-1}_{(opt)}}^{\omega^{Q-1}_{(ub)}} f_{Q-1}(\omega)\, d\omega + \cdots .$$

Table 3. Upper Bounds on the Transition Speeds for the Considered Example

s	$\omega^1_{(ub)}$	$\omega^2_{(ub)}$	$\omega^3_{(ub)}$	$\omega^4_{(ub)}$	$\omega^5_{(ub)}$	$\omega^6_{(ub)}$	$\mathbb{P}_{(ub)}$
s=6	6,500	6,043	4,848	3,676	2,996	1,637	3,504.84
s=8	6,500	4,285	3,629	2,996	1,871	1,214	2,753.8

Now, by composing every pair of contiguous integrals as

$$\int_{\omega^{j+1}_{(ub)}}^{\omega^{j}_{(opt)}} f_j(\omega)\, d\omega + \int_{\omega^{j}_{(opt)}}^{\omega^{j}_{(ub)}} f_j(\omega)\, d\omega = \int_{\omega^{j+1}_{(ub)}}^{\omega^{j}_{(ub)}} f_j(\omega)\, d\omega,$$

for all modes $j = 1, \ldots, Q$ (with $\omega^{Q+1}_{(ub)} = \omega^{min}$) it is possible to conclude that $\mathbb{P}_{(opt)} < \mathbb{P}_{(ub)}$. Hence, the theorem follows. □

Such an upper bound is fundamental to evaluate the "quality" of the algorithms proposed in this article, as it allows expressing a bound on the distance to the optimal solution. In other words, the distance of *any* solution to the upper bound $\mathbb{P}_{(ub)}$ provides a confidence interval which includes the actual (generally unknown) optimal solution.

4.2 Running Example

The algorithm shown in Figure 5 has been applied to the running example presented in Section 3.1 and the resulting switching speed upper bounds are shown in Table 3.

5 HEURISTIC APPROACHES

The non-convexity of the problem together with the lack of an analytical (closed-form) characterization of the schedulability constraint prevents the application of standard methods for computing the optimal solution.

A number of possible heuristic approaches can be devised to solve the problem. However, some of them are not immediately applicable, as explained below.

5.1 Gradient-Based Search

A first possible approach is to search the space of the switching speeds ω^j starting from a known feasible point and then increasing all the candidate transition speeds according to the gradient of the performance function $\nabla\mathbb{P}$. This is done by selecting a step δ and increasing each speed ω^j by $\delta \times p^j(\vec{\omega})$ until the boundary of the feasibility region is encountered. From that point on, the algorithm may proceed along the schedulability boundary with a local search. A local search consists in trying to improve as much as possible (using a binary search on the schedulability condition) all the transition speeds one by one, in order of their performance gradient $p^j(\vec{\omega})$.

This intuitive (and quite standard) approach does not work in many cases. This is because in most cases, the gradient of the performance function has higher components for lower transition speeds and the problem definition requires that $\omega^1 < \omega^2 < \cdots < \omega^M$. By projecting the desired direction onto this constraint, all the ω^j are increased by the same amount until we obtain the solution that computes the highest possible switching speed ω^1 for the first mode. This is often suboptimal. An example for a search space with only two switching speeds is reported in Figure 6, where the trajectory generated by such a heuristic is illustrated as a solid gray line that reaches the suboptimal solution $\vec{\omega_{so}}$. The gray curved lines in the figure join the set of points with the same performance value, while the black segmented line illustrates the (non-convex) feasibility boundary.

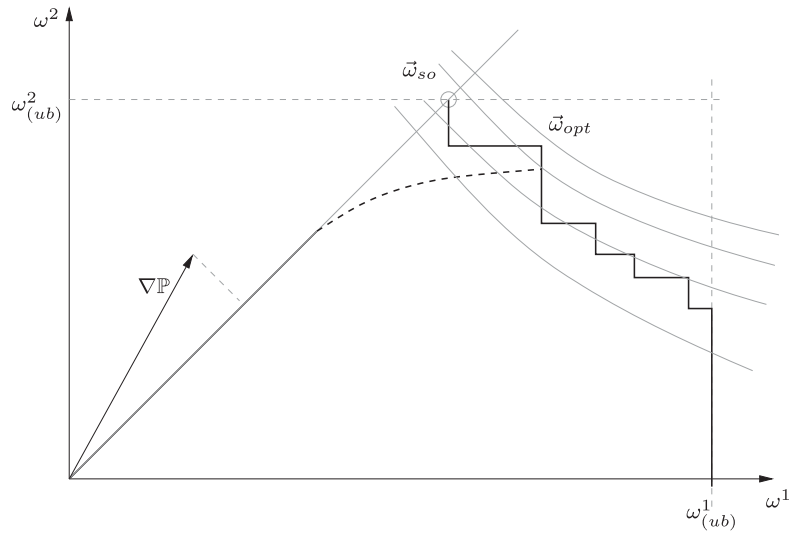

Fig. 6. Illustration for the modified gradient-based search in a bi-dimensional design space. The straight gray line illustrates the trajectory of a standard gradient-based search, while the dashed black line illustrates the corrected trajectory. The segmented black line depicts the (non-convex) feasibility region.

Indeed, this policy is similar to two other heuristics: a top-down or bottom-up greedy search. A bottom-up search (the top-down is similar) consists of finding the largest possible value of ω_1 that is feasible. Once ω_1 is determined, the algorithm tries to increase ω_2 to the largest possible amount that still results in a feasible system configuration and so on. Unfortunately, these methods are too greedy and often result in solutions in which most modes are simply not usable at all, and compute solutions that are far from optimality.

For example, in our running example, a top-down algorithm would produce $\omega^2 = w^2_{(ub)}$ and prevents the execution of the other control implementations Λ^j, with $j > 2$.

To improve results, the gradient-based heuristic can be corrected by a *penalty factor* associated to the gradient term for a given ω^j when the algorithm approaches its upper bound $\omega^j_{(ub)}$. An effective penalty function is

$$z^j(\omega) = 1 - e^{\left(\frac{\omega^j_{(ub)} - \omega}{\omega^j_{(ub)}}\right)^2}. \tag{12}$$

The trajectory component $T^j(\vec{\omega})$ for each switching speed w^j is expressed by

$$T^j(\vec{\omega}) = z^j(\omega^j) + \frac{p^j(\omega^j)}{p^{max}}, \tag{13}$$

where $p^{max} = \max_j p^j(\vec{\omega})$. Please note that the second term in $T^j(\omega)$ corresponds to the original gradient direction (defined in Equation (10)) with a normalization step of $\frac{1}{p^{max}}$.

Each switching speed ω^j is progressively increased with rate $\delta \times T^j(\vec{\omega})$ (for instance, $\delta = 5$ RPM has been used in our experiments), generating a sequence of speeds $\vec{\omega}(0), \vec{\omega}(1), \ldots, \vec{\omega}(k), \ldots$, in

```
1: procedure GradientSearch(C⃗)
2:     w^1(k) = ω^max, ∀k;
3:     for j = 2 to j = Q do
4:         ω^j(0) = ω^min;
5:     end for
6:     k ← 0;
7:     while Schedulable({(C^j, ω^j(k)), ∀j}) do
8:         for j = 2 to j = Q do
9:             ω^j(k + 1) ← ω^j(k) + δT^j(ω⃗(k));
10:        end for
11:        k ← k + 1;
12:    end while
13:    ω⃗_(end) ← LocalSearch(ω⃗(k − 1));
14:    return ω⃗_(end);
15: end procedure
```

Fig. 7. Pseudo-code for the proposed corrected gradient-based search.

Table 4. Results of the Application of the Corrected Gradient Heuristic to the Running Example

s	ω^1	ω^2	ω^3	ω^4	ω^5	ω^6	$\mathbb{P}$	
6	6,500	3,817	3,481	3,146	2,959	1,598	3,053.2	87.1% of $\mathbb{P}_{(ub)}$
8	6,500	1,460	1,419	1,366	1,361	1,168	1,934.2	70.2% of $\mathbb{P}_{(ub)}$

which each component progresses as

$$\begin{aligned} \omega^j(0) &= \omega^{min}, \\ \omega^j(k+1) = \omega^j(k) + \delta \times T^j(\vec{\omega}(k)), &\qquad \forall j = 2, \ldots, Q; \\ \omega^1(k) &= \omega^{max} \qquad \forall k. \end{aligned}$$

We hold $\omega^1(k) = \omega^{max}$ for the reasons explained in Section 4. The effect of the corrected gradient trajectory is shown in Figure 6 (dashed line).

From Equation (12), when approaching the upper-bound $\omega^j_{(ub)}$ for a mode j, the penalty term $z^j(\omega)$ is lowered (in an exponential manner) and reduces the increase rate for ω^j. This correction attempts at escaping from trivial local minima and improving the obtained solution.

Figure 7 illustrates the pseudo-code for the corrected gradient-based heuristic. The algorithm initializes the switching speeds at ω^{min} (except for the first mode). Then, a loop (line 7) updates the switching speeds according to the gradient trajectory $T^j(\vec{\omega})$ until the boundary of the schedulability constraint is reached. Finally, a local search is performed (see line 13) starting from the mode that has the maximum gradient coefficient, until no further steps can be performed without violating the schedulability constraint.

5.1.1 Running Example. The application of the corrected gradient heuristic to our running example provides the results reported in Table 4.

Clearly, the results in the second case are poor, since the modes after the first are all collapsed into a very small range and the final performance is far from the upper bound.

5.2 Utilization and Performance-Driven Backwards Search

An additional insight helps building a better heuristic: lowering a switching speed can give more freedom to the feasibility range of the others.

In particular, analyzing the experimental results of a more exhaustive branch and bound search (discussed in the next section), it became clear (as expected) that reducing the switching speed of the mode with the largest steady-state utilization provides more freedom for the other speeds, that is, allows one to keep them closer to their upper bound.

At the same time, the reduction of any transition speed ω^j causes a corresponding performance reduction that should be traded off with the utilization gain that favors the system schedulability. Both considerations are combined in the *Utilization and Performance-Driven backwards search*.

In this approach, the switching speeds are iteratively lowered, generating a sequence $\vec{\omega}(0)$, $\vec{\omega}(1), \ldots, \vec{\omega}(k), \ldots$, starting from the upper bound $\vec{\omega}(0) = \vec{\omega}_{(ub)}$. At each iteration, each ω^j is lowered by a quantity $\delta \times R^j$, that is,

$$\omega^j(k+1) = \omega^j(k) - \delta \times R^j(\vec{\omega}(k)), \qquad \forall j = 2, \ldots, Q \tag{14}$$

until a feasible set of speeds is found.

The reduction step $R^j(\vec{\omega})$ has been conceived for taking into account both the performance gradient and the steady-state utilization of the jth mode, thus obtaining

$$R^j(\vec{\omega}) = max(\widehat{U}^j + \widehat{P}^j(\vec{\omega}), R_{min}), \tag{15}$$

where R_{min} represents a minimum step to ensure progression (for instance, we used $R_{min} = 0.2$ RPM in our experiments). The terms $\widehat{U}^j$ and $\widehat{P}^j(\vec{\omega})$ are normalized indexes that relate to the utilization and performance gradient at the current switching speed $\omega^j(k)$. The index $\widehat{U}^j$ is defined as

$$\widehat{U}^j = \frac{U^j - U^{min}}{U^{max} - U^{min}}, \tag{16}$$

with

$$U^{max} = \max_j\{U^j\} \text{ and } U^{min} = \min_j\{U^j\}. \tag{17}$$

Similarly, $\widehat{P}^j(\vec{\omega})$ is computed as

$$\widehat{P}^j(\vec{\omega}) = \frac{p^{max} - p^j(\vec{\omega})}{p^{max} - p^{min}}, \tag{18}$$

where $p^{max} = \max_j p^j(\vec{\omega})$ and $p^{min} = \min_j p^j(\vec{\omega})$.

Note that, to match the objective of this heuristic, the larger the steady-state utilization of a mode the larger the corresponding index $\widehat{U}^j$. Analogously, the larger the performance gradient, the lower the index $\widehat{P}^j(\vec{\omega})$.

Figure 8 illustrates the pseudo-code for the proposed backwards search. The algorithm lowers each switching speed ω^j until the system becomes schedulable (see line 7), then it proceeds with a local search, as the gradient-based search of Section 5.1.

5.2.1 Running Example. The application of the backwards search heuristic to our running example provides the results reported in Table 5.

The backwards search heuristic is very effective. The running example is not a special case, as shown in our experimental results (Section 7). The computed values are always very close to the optimum. Also, compared with the previous heuristic, a set of higher transition speeds for lower index modes is traded for a lower transition speed for the last mode (which results in a better performance anyway).

```
1: procedure BACKWARDSSEARCH(C⃗)
2:     w^1(k) = ω^max, ∀k;
3:     for j = 2 to j = Q do
4:         ω^j(0) = ω^j_ub;
5:     end for
6:     k ← 0;
7:     while NOT SCHEDULABLE({(C^j, ω^j(k)), ∀j}) do
8:         for j = 2 to j = Q do
9:             ω^j(k + 1) ← ω^j(k) − δR^j(ω⃗(k));
10:        end for
11:        k ← k + 1;
12:    end while
13:    ω⃗_(end) ← LOCALSEARCH(ω⃗(k));
14:    return ω⃗_(end);
15: end procedure
```

Fig. 8. Pseudo-code for the proposed utilization and performance-driven backwards search.

Table 5. Results of the Application of the Backwards Search Heuristic to the Running Example

s	ω^1	ω^2	ω^3	ω^4	ω^5	ω^6	$\mathbb{P}$	
6	6,500	6,039	4,836	3,672	2,899	1,630	3,480.2	99.3% of $\mathbb{P}_{(ub)}$
8	6,500	4,282	3,194	2,887	1,868	1,050	2,644.0	96.0% of $\mathbb{P}_{(ub)}$

6 BRANCH AND BOUND

In many cases, the optimum can be computed (within a given speed granularity) by performing an exhaustive search starting from an initial feasible solution and attempting to extend each transition speed toward its upper bound. The knowledge of performance upper and lower bounds allows introducing effective pruning rules that stop the algorithm when a solution with sufficient quality is obtained.

6.1 Definition of the Algorithm

The algorithm makes use of speed upper bounds to compute a performance upper bound $\mathbb{P}_{ub}$ that is, in general, not feasible. Also, one of the heuristics in the previous section (the backwards search is the best option) allows computing a lower bound on the optimum performance $\mathbb{P}_{lb}$.

The algorithm requires the definition of a speed resolution δ (in our experiments δ = 15 RPM) and a starting feasible solution with a configuration of transition speeds that has ω^Q at the highest possible value that allows for the execution of the other modes. The starting solution should also be *maximal*, meaning that any possible increase of any transition speed would create a non-feasible solution. Given any solution, a maximal solution can be simply found by a local search. Because of the monotonicity property of our performance function (on Λ^i and C^i), a maximal solution is always going to have higher performance than any solution for which the transition speeds are component-wise less than or equal.

The search algorithm is based on the observation that, given a maximal solution, any increase in a transition speed ω^j can be obtained by decreasing at least one of the transition speeds ω^k with $k > j$. The algorithm works iteratively, attempting to improve on an initial feasible solution $\vec{\omega}_s$ with ω_s^Q equal to the largest possible value that allows for the execution of the other modes (the algorithm to compute the initial solution is explained later).

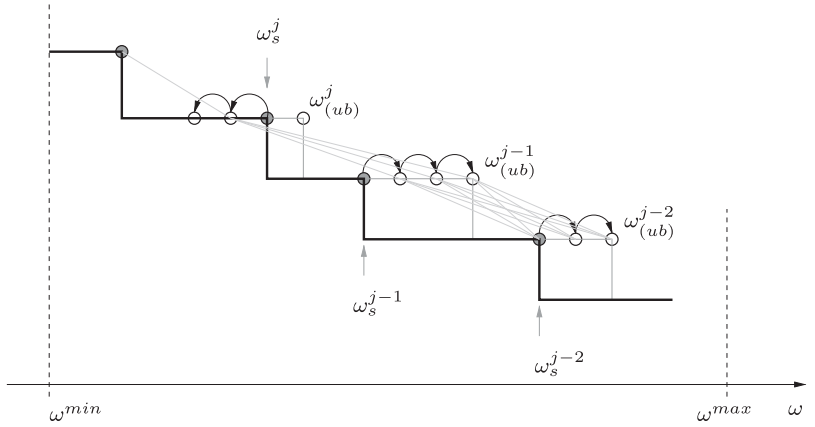

Fig. 9. The optimization algorithm as a branch and bound search in the domain of the ω.

At each iteration with index j (j goes from Q to 3), the values in the set $\{\omega_s^Q, \omega_s^{Q-1}, \ldots, \omega_s^{j+1}\}$ are left unchanged from $\vec{\omega}_s$ (the set is empty when $j = Q$): this is because the algorithm is similar to a depth-first search in a tree, where the sub-tree corresponding to modes with index $< j$ is explored.

The speed ω_s^j is then iteratively reduced by $m_j \times \delta$, with $m_j \in \mathbb{N}^+$, and for each value of m_j the algorithm tries all the possible extensions (as integer multiples of δ) of the speeds $\{\omega_s^{j-1}, \ldots, \omega_s^2\}$, until it reaches the feasibility boundary (i.e., a *maximal* solution within the δ resolution). As a result, the algorithm performs a branch and bound search on the tree of speeds with index lower than j. Figure 9 shows an intermediate step of the algorithm with the corresponding search tree below the element with index j. Since the index j is progressively lowered until 3, all the possible speed combinations (with granularity δ) are tried. The index j (controlling the switching speed that is selectively reduced) ends in 3 because it is always $\omega^1 = \omega^{max}$ and reducing ω^2 (starting from the initial maximal solution) is pointless because ω^1 cannot be further increased.

Computing the initial solution $\vec{\omega}_s$. The solution $\vec{\omega}_s$ is computed iteratively. First, the value of ω_s^Q is computed by searching back from $\omega_{(ub)}^Q$ (using a binary search) until the largest value that allows the execution of all other modes with lower index j at transition speeds $\omega_s^Q + \epsilon \times (Q - j)$ (with ϵ arbitrarily small). When the algorithm moves to the next mode $Q - 1$, ω_s^{Q-1} is similarly computed as the largest speed that allows executing all other modes with index $j < Q - 1$ with a transition in $\omega_s^{Q-1} + \epsilon \times (Q - 1 - j)$ and so on.

6.2 Pseudo-Code

Figure 10 illustrates the pseudo-code for the proposed branch and bound algorithm. In addition to the WCETs of the available control implementations, the algorithm takes as input a lower bound on the overall performance $\mathbb{P}_{(lb)}$, which can be obtained by executing one of the heuristics presented in the previous section.

At any point in time, the search algorithm keeps track of the best performance solution $\mathbb{P}_{(cur)}$ found until then, which is initialized to the performance lower bound (line 2 in Figure 10). After initializing the switching speeds at their upper bound (line 3), the algorithm calls the SEARCH recursive sub-procedure to search for the optimal solution. The search starts from the switching speed of the Q-th mode and explores the tree of possible values for the switching speeds of all the modes (within the given granularity δ). The search tree has Q levels, one for each mode. Every

```
1: procedure BranchAndBound(C⃗, ℙ_(lb))
2:     ℙ_(cur) ← ℙ_(lb);
3:     ω⃗_(opt) ← ω⃗_(ub);
4:     Search(C⃗, Q);
5:     return ω⃗_(opt);
6: end procedure
7:
8: procedure Search(C⃗, j)
9:     ω_s^j ← InitialSolution(j);
10:    if j > 2 then
11:        ω^j_(lb) ← ω^(j+1) + ϵ;
12:        for ω^j = ω_s^j to ω^j = ω^j_(lb) with step −δ do
13:            ℙ_(max) ← ℙ(ω^1_(ub), ..., ω^(j−1)_(ub), ω^j, ..., ω^Q);
14:            if ℙ_(max) > ℙ_(cur) then
15:                Search(C⃗, j − 1);
16:            end if
17:        end for
18:    else
19:        if ℙ(ω^1, ..., ω^Q) > ℙ_(cur) then
20:            ℙ_(cur) ← ℙ(ω^1, ..., ω^Q);
21:            ω⃗_(opt) ← ω⃗;
22:        end if
23:    end if
24:    return ω⃗_(opt);
25: end procedure
```

Fig. 10. Pseudo-code for the branch and bound algorithm.

instance of the SEARCH procedure, invoked with index j, is in charge of exploring the sub-tree at levels $< j$ by starting from the initial solution for the j-th switching speed (which is computed as discussed above).

The procedure uses of a conditional statement to distinguish between (i) the recursive step and (ii) the stop condition of the recursive algorithm.

The recursive step consists in opening a set of branches to explore the sub-tree, each corresponding to a reduced value of the j-th switching speed. The switching speed ω^j is progressively reduced by δ: the reduction starts from the initial solution ω_s^j and ends when the switching speed of the $(j+1)$-th mode is reached (line 12).

Not all the branches are actually explored. The algorithm performs a *pruning* on the sub-tree when the current best performance value cannot be improved by *any* solution available in the sub-tree (line 14). The performance of *all* the solutions available in the current sub-tree are upper bound by the performance of the set of speeds $\{\omega^1, \omega^2_{(ub)}, \ldots, \omega^{j-1}_{(ub)}, \omega^j, \omega^{j+1}, \ldots, \omega^Q\}$ (similarly as stated by Theorem 4.1), which is constructed by leveraging on the knowledge of the speed upper bounds.

The stop condition holds when the search reaches the second level, i.e., $j = 2$ (remember that ω^1 is fixed and equal to the maximum engine speed). In this case, the procedure has traversed the entire search tree, hence a particular solution exists. If such a solution improves the best performance that has been found so far (line 19), then it is stored as optimal (line 21).

6.3 Discussion

The execution of the algorithm showed how the optimum performance often results in a configuration in which the largest speed decrease is for the mode with highest local utilization. This

Table 6. Results for the Running Example with $s = 8$ Applying All the Algorithms Presented in this Article

Algorithm	ω^1	ω^2	ω^3	ω^4	ω^5	ω^6	$\mathbb{P}$
Gradient-based search	6,500	1,460	1,419	1,366	1,361	1,168	1,934.2 70.2% of $\mathbb{P}_{(ub)}$
Backwards search	6,500	4,282	3,194	2,887	1,868	1,050	2,644.0 96.0% of $\mathbb{P}_{(ub)}$
Branch and bound	6,500	4,274	3,556	2,778	1,858	1,044	2,665.9 96.8% $\mathbb{P}_{(ub)}$
Upper bound	6,500	4,285	3,629	2,996	1,871	1,214	$\mathbb{P}_{(ub)} = 2753.8$

was the motivation for deriving the utilization-driven backwards search heuristics presented in Section 5.2.

The branch and bound computes solutions of very good quality at the expense of time. A set of experiments (see Section 7.3) has been performed to evaluate the execution times and how the branch and bound results compare with respect to the results from the heuristics. However, it should be noted that the runtime of the branch and bound search is heavily dependent on the performance lower bound $\mathbb{P}_{lb}$ that is provided to prune the solution tree at the beginning. This value is obtained by the backwards search heuristic. Hence, even in those cases in which the problem can be solved to (almost) optimality by the branch and bound search, a practically usable execution time can only be achieved thanks to the availability of a very good (and fast) heuristic.

6.4 Running Example

The availability of the branch and bound exhaustive search for the optimum (with finite granularity) allows an evaluation of the quality of the heuristics. Table 6 shows a summary of the results for the case with $s = 8$.

The table shows the typical result found in our experiments. Not only is the backwards heuristic very close to the upper bound, it is also extremely close to the value computed by the branch and bound search. In reality, the branch and bound result is much closer to the result of the heuristic than it is to the upper bound.

7 EXPERIMENTAL RESULTS

This section reports a set of experimental results aimed at evaluating and comparing the approaches presented in this article. All the algorithms have been implemented in the C++ language and tested over synthetic workload for measuring their effectiveness.

In the experiments, the speed limits of the engine have been set to $\omega^{min} = 500$ RPM and $\omega^{max} = 6500$ RPM, respectively (typical values for a production car). The acceleration range allows the engine to reach the maximum speed starting from the minimum in 35 revolutions (Davis et al. 2014), resulting in $\alpha^+ = -\alpha^- = 1.62 \cdot 10^{-4}$ rev/msec^2.

7.1 Workload Generation

In the evaluation, we consider a task set composed of N periodic tasks, with utilization U^P, and an AVR task τ^* with $Q = 6$ possible control implementations.

The periods of the periodic tasks are $\{5, 10, 20, 50, 80, 100\}$ms, considered as typical values for engine control applications (Guzzella and Onder 2010). The execution times of the periodic tasks are generated by the UUnifast algorithm (Bini and Buttazzo 2005). The WCETs of the possible control implementations for the AVR task are generated by randomly choosing (with a uniform distribution and a minimum separation c^{sep}) a set of *seed values* $\{c^1, c^2, \ldots, c^Q\}$ from the range $[c^{min}, c^{max}]$. The actual WCETs are computed using a *scale factor* s as $C^j = s \cdot c^j$. The scale factor is

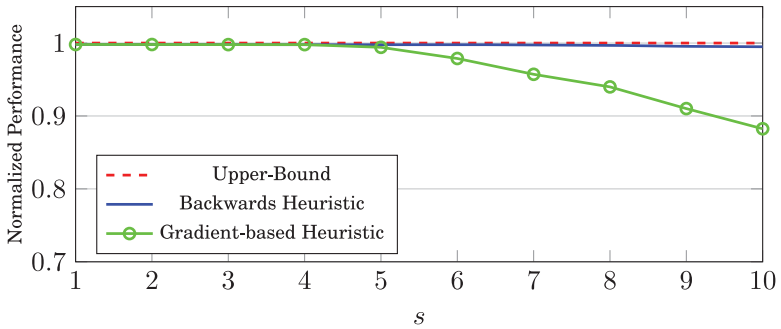

Fig. 11. Performance of the two heuristics (Section 5) as a function of s for $U^P = 0.5$. The results are normalized to the performance upper-bound.

a parameter that allows tuning the computational requirements of the AVR task implementations. When the switching speeds and consequently the interarrival times of the AVR task are unknown, it is not possible to define the AVR load with a simple utilization metric.

7.2 Performance Functions Generation

The performance functions considered in this work are (i) constant functions, as in Equation (5), and (ii) exponential functions of the engine speed, as in Equation (7). In the first case, each control implementation Λ^j is assigned a performance coefficient k^j that is randomly generated with a uniform distribution in the range $[k^{min}, k^{max}]$, with a minimum separation of k^{sep}. In the case of exponential functions, the generation involves two parameters $k^{j,1}$ and $k^{j,2}$ for each control implementation Λ^j. The performance is normalized with respect to Λ^Q (i.e., the implementation with the largest WCET with constant performance), which has $k^{Q,1} = 1$ and $k^{Q,2} = 0$. To provide for a uniform distribution of the exponential performance functions (see Equation (6)), the coefficients $k^{j,2}$ are generated with a uniform distribution in a logarithmic scale with range $[\log k^{2,min}, \log k^{2,max}]$. Finally, we set $k^{j,1} = 1, j = 1, \ldots, Q$ for simplicity.

7.3 Constant Performance Functions

In this experiment, we consider the constant performance functions and evaluated the performance of the heuristics with respect to the upper-bound $\mathbb{P}_{(ub)}$ obtained with the method described in Section 4. We generate 500 task sets with $N = 5$ periodic tasks and an AVR task with a set of possible control implementations. For each task set, we tested 30 different sets of performance coefficients and a variable scale factor s from 1 to 10, trying 150,000 different configurations in total.

For each value of s the performance of the heuristics and the upper-bound are computed. The performance values obtained by the heuristics are normalized with respect to the value of the upper-bound. As a consequence, the obtained values are lower-bounds of the performance values normalized with respect to the actual optimal performance. The normalized performance values were then averaged among all the configurations for a given value of s.

The range and separation $c^{min} = 100$, $c^{max} = 1,000$, $c^{sep} = 100$ are used for generating the computation time seeds and $k^{min} = 1$, $k^{max} = 50$, and $k^{sep} = 1$ are used for the generation of the performance coefficients.

Figure 11 shows the results for the case of utilization $U^P = 0.5$ of the periodic tasks. As shown by the graph, the backwards heuristic provides an extremely good performance, always greater

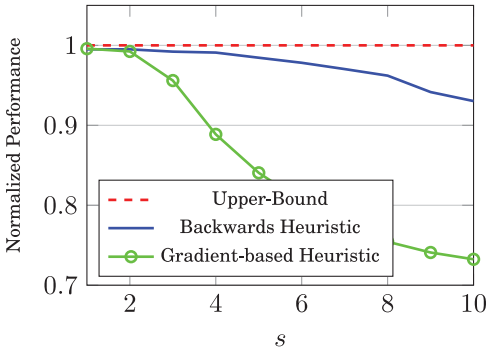

Fig. 12. Performance of the two heuristics as a function of s for $U^P = 0.75$. The results are normalized to the performance upper-bound.

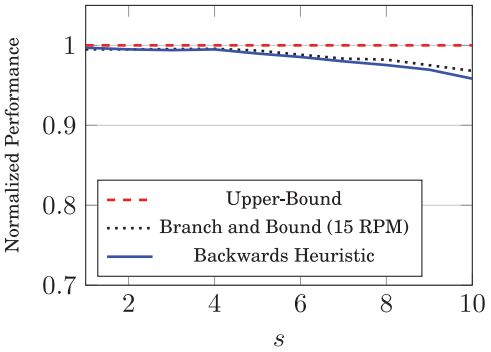

Fig. 13. Performance of the backwards heuristic and the branch and bound algorithm as a function of s for $U^P = 0.75$. The results are normalized to the performance upper-bound.

than 99% of the upper-bound, and extremely close to the optimum. Conversely, the gradient-based heuristic shows a degradation for increasing values of s reaching a value lower than the 90% of the upper-bound for $s = 10$. In our experiments, the gradient-based heuristic always performs worse than the backwards search.

Figure 12 reports the results for the same experiment when the utilization of the periodic tasks is increased to $U^P = 0.75$. The performance of the backwards search heuristic is slightly worse, reaching a value of approximately 93% for $s = 10$, while the gradient-based heuristic shows a consistent degradation for $s > 3$.

However, as explained at the beginning of this section, the results normalized with respect to the upper-bound are only lower-bounds of the actual performance, expressed by the ratio with respect to the true optimum performance value (when computable). For this reason, we performed another set of experiments including the result of the branch and bound algorithm (with δ =15 RPM), to study the performance of the backwards search heuristic with respect to the actual optimal performance (or a value most likely close to it). Due to the large runtime of the branch and bound algorithm, this experiment has been conducted on a small set of configurations with 50 task sets and 5 sets of performance coefficients. The results are shown in Figure 13. As shown by the graph, the optimal performance tends to recede from the upper-bound for increasing values of s, confirming the effectiveness of the backwards search heuristic which remains around 99% of the performance value found by the branch and bound algorithm.

Running Times. The maximum observed running time for the backwards search heuristic is 756s with an average runtime of 5.6s. The maximum observed running time for the gradient-based heuristic is 999s with an average runtime of 182s. For the branch and bound algorithm with precision δ =15 RPM we measured a maximum runtime of 16,070s with an average runtime of about 600s. Such results have been obtained executing the algorithms on a machine equipped with an Intel i7 processor running at 3.2GHz and 8Gb of RAM.

7.4 Exponential Performance Functions

Another experiment has been conducted with the exponential performance functions described in Section 3. We focus on the comparison of the two heuristics against the performance upper-bound $\mathbb{P}_{(ub)}$.

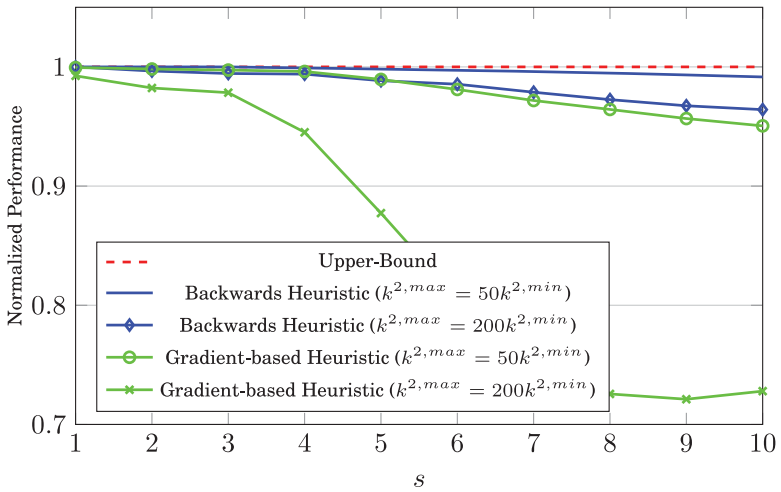

Fig. 14. Performance of the two heuristics as a function of s for $U^P = 0.75$. Exponential performance functions under two different configurations are considered. The results are normalized to the performance upper-bound.

Figure 14 shows the results as a function of the scale factor s for two different values of the coefficient $k^{2,max}$ of the performance functions (keeping $k^{2,min}$ constant). The utilization of the periodic tasks is $U^P = 0.75$ and for each value of s we try 500 task sets and 30 sets of performance coefficients, hence testing 150,000 different configurations.

As shown by the graph, the backwards search heuristic has a performance always greater than the 99% of the upper-bound for $k^{2,max} = 50k^{2,min}$. In the same setting, the gradient-based heuristic shows a slight performance degradation, reaching the 95% of the upper-bound for $s = 10$. In the case $k^{2,max} = 200k^{2,min}$, there is a slight degradation also for the backwards heuristic, which reaches the 96% of the upper-bound for $s = 10$, while the gradient-based heuristic shows a consistent degradation for $s > 4$.

The ratio between $k^{2,max}$ and $k^{2,min}$ determines the distribution of the exponential performance functions in the speed domain. Intuitively, the higher $k^{2,max}/k^{2,min}$ the more the performance functions are far apart. For this reason, we conduct another experiment by varying the ratio $k^{2,max}/k^{2,min}$ while holding the scale factor $s = 7$.

For each value of the ratio we test 500 task sets and 50 sets of performance coefficients $k^{2,j}$, hence 500,000 configurations. The results are shown in Figure 15 and confirm the trend of Figure 14, showing a graceful and quite limited degradation of the performance of the backwards heuristic for increasing values of the ratio $k^{2,max}/k^{2,min}$. Again, the gradient-based heuristic shows worse performance, reaching very low values for $k^{2,max}/k^{2,min} > 200$.

8 VALIDATION OF THE PERFORMANCE MODEL

The performance model presented in Section 3 has been validated using a *simulation framework*, which has been developed to model and execute the main components of the considered CPS: the engine, the control logic, and the task implementation. In the context of this work, the simulation framework has been used to compute the performance functions under different fuel injection strategies and verify our fitting hypothesis to the family of exponential functions expressed in Equation (6). The framework has also been used to explore the dependency of the performance with respect to different control implementations.

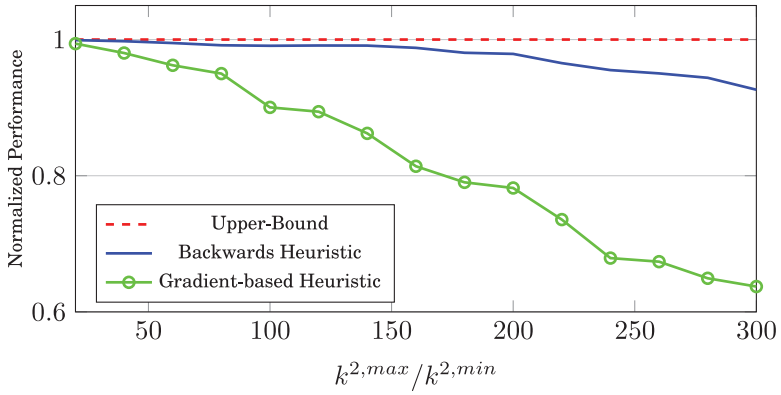

Fig. 15. Performance of the two heuristics as a function of $\frac{k^{2,max}}{k^{2,min}}$ for $U^P = 0.75$ and $s = 7$. The results are normalized to the performance upper-bound.

8.1 Simulation Framework

The system model has been created in the MATLAB Simulink ® environment by integrating three major subsystems. The *engine model* and the model of the *injection control logic* have been developed as traditional Simulink systems, with blocks encoding the differential equations of the engine dynamics and the control algorithms (including the tables for determining the injection phase and duration).

The implementation of the controls as a set of concurrent tasks with variable-rate activation and an adaptive behavior (and execution time) has been modeled using the T-Res framework (Cremona et al. 2015), specifically developed for studying the impact of task scheduling on control performance and purposely extended to add the representation of AVR tasks.

8.2 Engine Modeling

Modeling a diesel engine is a challenging problem due to the multiple complex physical parts to be considered, obtaining a *highly non-linear* dynamic system with at least one dynamic feedback loop for the turbocharger.

The engine model developed for this work is an abstraction of a modern four-cylinder heavy-duty diesel engine, equipped with a common-rail injection system and a Variable Geometry Turbocharger (VGT). The model includes a set of physical equations describing the engine dynamics, taken from the specialized literature, some functions defining subsystem models based on experimental data, and *static maps* (e.g., for the turbocharger model).

The main sources used for modeling the air path are the books by Guzzella and Onder (2010), Panse (2005), and Criens (2014)). A simplified model of combustion has been created following the description in Ding (2011) and Ericson et al. (2006) to simulate the behavior of pressure, temperature, emissions of Nitrogen Oxide (NO_2) and thermodynamic efficiency. The equations that describe the turbocharger behavior have been taken from the recent book by Nguyen-Schäfer (2015).

8.3 Engine Control

In agreement with the current industrial practice, the control of the engine is based on a mix of static *maps* and additive control loops to correct possible deviations from the set points. This strategy is a typical solution for commercial cars, where maps are calibrated on *test benches*, in

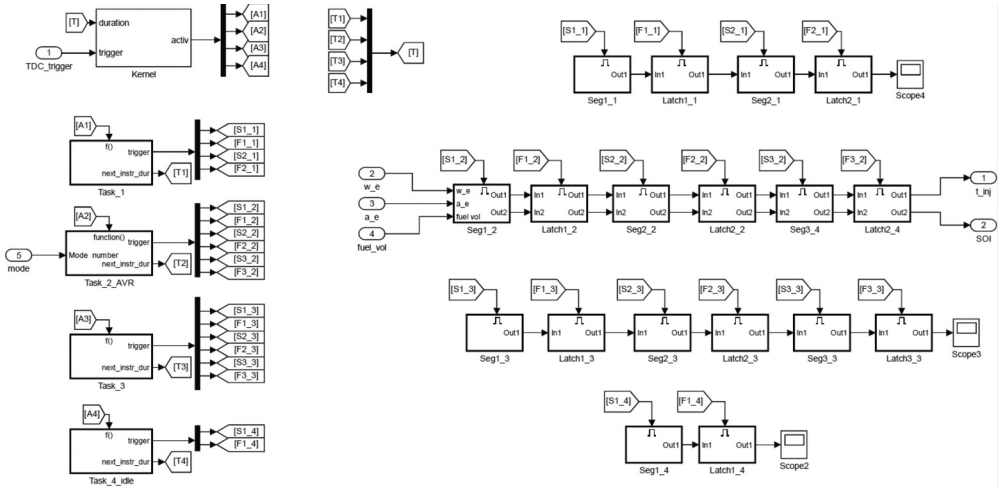

Fig. 16. Injection strategies used in the simulation: from the top to the bottom, triple injection, double injection, and single injection.

optimal conditions, and additional loops are added to face deviations from the optimal behavior due to transients, aging, and changes of external conditions.

Our engine control functions include the typical injection profile for diesel engines, consisting of (i) an optional number of *pre-injections*, (ii) a central *main injection*, and (iii) one or two optional *post-injections*. Pre-injections are used to heat the combustion chamber and ensure a more uniform fuel-air mixture for the main injection, while post-injections are typically used to burn the residual and decrease the amount of pollutants (Criens 2014; Bhatt et al. 2013).

8.4 Control Task Model

The task set in the engine model considers a single AVR task computing all the functions related to the fuel injection control. The focus of the modeling is to understand how the performance functions of the system change as the system switches through the different modes implemented in the AVR task.

For the purpose of this work, the task execution modes are modeled according to three different injection strategies: triple, double, and single injection. Each strategy corresponds to a mode of the AVR task (with decreasing WCET).

Figure 16 shows the set of Simulink subsystems representing the control tasks, the scheduler, and the functions executed by the tasks. The custom blocks for the representation of the scheduler and the tasks are described in detail in Cremona et al. (2015). The block on the top-left corner of Figure 16 contains the model of the fixed-priority scheduler. The blocks below it are the control tasks. Three of them are periodic; the second from the top is the AVR task controlling the fuel injection. With respect to the periodic task blocks, the AVR task has an additional input defining the execution mode (and its corresponding WCET). The chains of blocks on the right side of Figure 16 are the subsystems executed by the tasks. The second chain from the top is the sequence of the control subsystems executed by the AVR task.

The three AVR modes lead to different heat release profiles, thus directly influencing the combustion dynamics. Typically, multiple injections decrease the efficiency of torque generation, but also decrease the pollutant emissions, by lowering the in-cylinder peak temperature and improving the fuel combustion with pre-heating and better air-fuel mixing (Criens 2014; Bhatt et al. 2013).

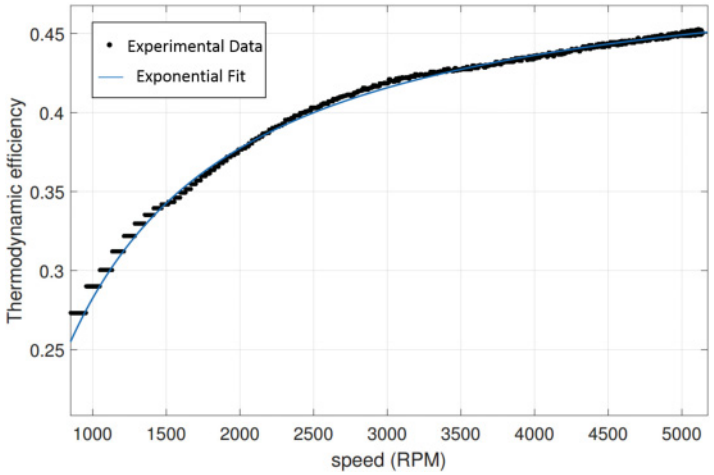

Fig. 17. Experimental data for the thermodynamic efficiency as a function of the engine speed. The continuous line illustrates a fit to the exponential performance model proposed in this work.

8.5 Experimental Evaluation

The objectives of the simulation are to characterize the performance functions and model their dependency on the engine speed. The performance functions available with the current model setting are (i) the *thermodynamic efficiency* and (ii) the *emissions* of NO_2. The simulation runs have been defined for a model input with low external torque (no gear subsystem is included) and a throttle input profile with a sudden acceleration (maximum throttle) of a duration of approximately 2s. The data obtained from the simulation have been studied as a function of the engine speed (ranging from 500 to 5,500 RPM).

For the three different injection strategies (one to three injections), the simulation results show that the performance functions can be fitted well with the exponential behavior reported in Equation (6). The fittings have been made using the *Curve Fitting Tool* of MATLAB, with the LAR Trust-Region algorithm.

The experimental data for the thermodynamic efficiency are reported in Figure 17, where also the fit to our exponential performance model is reported as a continuous line.

As clear from the graph, the fit is very accurate and resulted in a coefficient of determination (R squared) of 0.998 (1 means a perfect fit) when Equation (6) is configured with $k_1 = 0.5042$ and $k_2 = 579.2$.

Because of the strong dependency on the temperature, the emissions of NO_2 measured with the simulated framework resulted quite noisy—with particular sensitivity at little variations for values greater than 1700°K. However, as shown in the example illustrated in Figure 18, the behavior appeared clearly correlated with an exponential shape. In fact, the fit to Equation (6) (illustrated as a continuous line in the figure) resulted in a coefficient of determination equal to 0.94 when configured with $k_1 = 2,619$ and $k_2 = 1.625 \cdot 10^4$.

8.6 Comparing Performance Functions with Different AVR Modalities

With the same setting and approach described in the previous section, we also evaluated the dependency of the performance functions on the mode of the AVR task, which in our case corresponds to a given injection strategy. For each mode, the performance has been first measured as a function

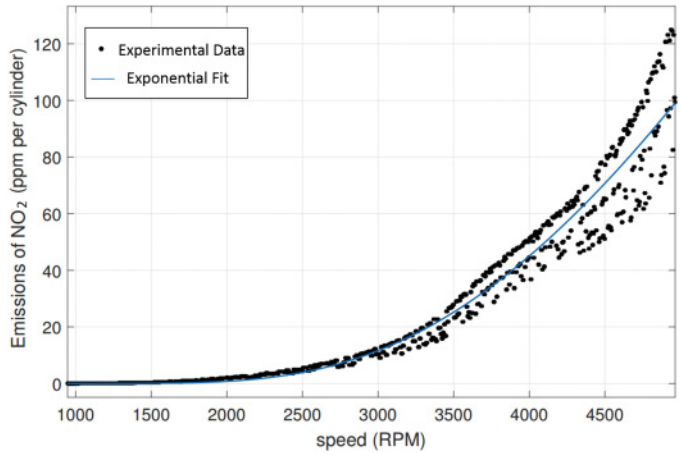

Fig. 18. Experimental data for the emissions of NO_2 as a function of the engine speed. The continuous line illustrates a fit to the exponential performance model proposed in this work.

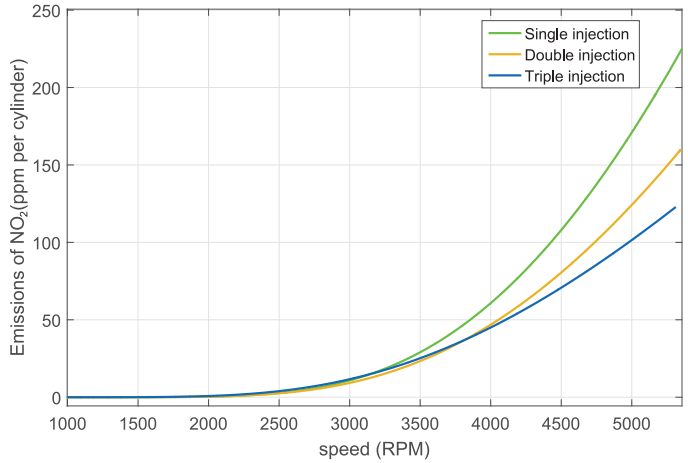

Fig. 19. Fit to the exponential performance model of the NO_2 emissions for different injection strategies.

of the engine speed and then fit to the proposed exponential model. Figure 19 reports the results of this experiment for the NO_2 emissions. As can be observed from the graph, the performance functions match our monotonicity assumption with the complexity of the control implementation (a control for the triple injection strategy requires more computation than the one with double injection, and so on).

9 RELATED WORK

A model for describing an AVR task was first proposed by Kim, Lakshmanan, and Rajkumar (Kim et al. 2012), who also derived a schedulability analysis under very restrictive assumptions, considering a single AVR task running at the highest priority with a period always smaller than those of the other tasks. In addition, relative deadlines were assumed to be equal to periods and priorities were assigned based on the rate-monotonic algorithm. Pollex et al. (2013) derived a sufficient feasibility test for fixed-priority scheduling, but assuming a constant engine speed. Davis et al. (2014)

used an Integer Linear Programming (ILP) formulation to derive a sufficient schedulability test of AVR tasks under fixed priorities, also taking acceleration into account, but considering a finite set of (discretized) initial engine speeds.

The exact characterization of the interference produced by an AVR task under fixed priorities has been presented by Biondi et al. (2014) as a search approach in the speed domain, where the concept of dominant speeds is used to reduce the complexity and avoid speed quantization. Such a method has then been extended to derive an exact response time analysis of fixed priorities AVR tasks (Biondi et al. 2015).

Other works addressed the analysis of AVR tasks under the EDF scheduling algorithm. Guo and Baruah (2015) proposed a speedup factor analysis and sufficient schedulability tests for AVR tasks scheduled with EDF. Biondi et al. (2015) proposed a precise workload characterization generated by an AVR task that is used to derive a feasibility analysis under EDF scheduling.

10 CONCLUSIONS

The problem of performance-oriented design of transition speeds in a fuel injection system with adaptive variable rate tasks has been discussed. A set of optimization algorithms has been presented, assuming a quite general scenario in which the performance of each control implementation is expressed by an arbitrary function that has the only requirement of being integrable and monotonically increasing with the complexity of the implemented algorithm.

The experimental results show that the proposed heuristics are quite close to the actual optimal value and allow the computation of the optimum with finite resolution in many cases.

A set of Simulink models, including the CPS model of the engine, the fuel injection system, the control functions, and their task implementation (with variable execution times) have been defined to support the assumptions on which our algorithms are based (fitting the engine performance functions with exponential curves) and study the dependency of the control performance with respect to a task implementation. Future work includes an experimental analysis and validation with actual test bench data and extending the proposed optimization method to schedulability constraints that allow for temporary overload conditions.

ACKNOWLEDGMENTS

The authors would like to thank Pasquale Buonocunto for his valuable help and fruitful discussions for setting up the engine model and the engine controllers.

REFERENCES

N. M. Bhatt, P. P. Rathod, A. S. Sorathiya, and Ramesh P. 2013. Effect of multiple injections on the performance and emission of diesel engine-A review study. *International Journal of Emerging Technology and Advanced Engineering* 3, 3 (2013), 134–140.

Enrico Bini and Giorgio C. Buttazzo. 2005. Measuring the performance of schedulability tests. *Real-Time Systems* 30, 1–2 (2005), 129–154.

Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. 2008. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems* 39, 1–3 (August 2008), 5–30.

Alessandro Biondi, Giorgio Buttazzo, and Stefano Simoncelli. 2015. Feasibility analysis of engine control tasks under EDF scheduling. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15)*.

Alessandro Biondi, Alessandra Melani, Mauro Marinoni, Marco Di Natale, and Giorgio Buttazzo. 2014. Exact interference of adaptive variable-rate tasks under fixed-priority scheduling. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS'14)*.

Alessandro Biondi, Marco Di Natale, and Giorgio Buttazzo. 2015. Response-time analysis for real-time tasks in engine control applications. In *Proceedings of the 6th International Conference on Cyber-Physical Systems (ICCPS'15)*.

Alessandro Biondi, Marco Di Natale, and Giorgio Buttazzo. 2016. Performance-driven design of engine control tasks. In *Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS'16)*.

Alan Burns and Sanjoy Baruah. 2008. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering* 2, 1 (2008), 74–97.

Giorgio Buttazzo, Enrico Bini, and Darren Buttle. 2014. Rate-adaptive tasks: Model, analysis, and design issues. In *Proceedings of the International Conference on Design, Automation and Test in Europe*.

Darren Buttle. 2012. Real-time in the prime-time. In *Keynote Speech at the 24th Euromicro Conference on Real-Time Systems*.

F. Cremona, M. Morelli, and M. Di Natale. 2015. TRES: A modular representation of schedulers, tasks, and messages to control simulations in Simulink. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 1940–1947.

CHA Criens. 2014. *Air-Path Control of Clean Diesel Engines: For Disturbance Rejection on NOx, PM and Fuel Efficiency*. Ph.D. Dissertation. Technische Universiteit Eindhoven.

Robert I. Davis, Timo Feld, Victor Pollex, and Frank Slomka. 2014. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium*.

Yu Ding. 2011. Characterising combustion in diesel engines. *TU Delft* 668 (2011).

Claes Ericson, Björn Westerberg, Magnus Andersson, and Rolf Egnell. 2006. *Modelling Diesel Engine Combustion and NOx Formation for Model Based Control and Simulation of Engine and Exhaust Aftertreatment Systems*. Technical Report. SAE Technical Paper.

Zhishan Guo and Sanjoy Baruah. April 2015. Uniprocessor EDF scheduling of AVR task systems. In *Proceedings of the ACM/IEEE 6th International Conference on Cyber-Physical Systems (ICCPS'15)*.

Lino Guzzella and Christopher H. Onder. 2010. *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer-Verlag.

INTERESTED, European project, cordis project description. Retrieved from http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/factsheets/inte rested.pdf.

Junsung Kim, Karthik Lakshmanan, and Ragunathan Rajkumar. 2012. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *Proceedings of the 3rd IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS'12)*. 28–38.

Hung Nguyen-Schäfer. 2015. *Rotordynamics of Automotive Turbochargers*. Springer.

Pushkaraj A. Panse. 2005. *Dynamic Modeling and Control of Port Fuel Injection Engines*. Ph.D. Dissertation. Citeseer.

Victor Pollex, Timo Feld, Frank Slomka, Ulrich Margull, Ralph Mader, and Gerhard Wirrer. 2013. Sufficient real-time analysis for an engine control unit with constant angular velocities. In *Proceedings of the Design, Automation and Test Conference in Europe*.

Received October 2016; revised May 2017; accepted July 2017