Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS 2016), Vienna, Austria, April 11-14, 2016.

Performance-driven Design of Engine Control Tasks

Alessandro Biondi, Marco Di Natale, Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {alessandro.biondi, marco, giorgio}@sssup.it

***Abstract*—Engine control tasks include computational activities triggered at specific rotation angles of the crankshaft, making the computational load increase with the engine speed. To avoid overload at high speeds, simplified control implementations are used, defining different operational modes at different speed intervals. The design of a set of adaptive variable rate tasks is an optimization problem, consisting in determining the rotation speeds at which mode changes should occur to optimize the system performance while guaranteeing the schedulability. This paper presents three methods for tackling the optimization problem under a set of assumptions about the performance metric and the problem constraints. Two are heuristics and one is a branch and bound that is guaranteed, when it terminates, to find the optimum within a given granularity. In addition, a simple method to compute a performance upper bound is presented. The analysis of the problem reveals several insights for the design and the heuristics are shown to be quite close to the performance upper bound and the optimum with finite granularity.**

I. Introduction

Engine control is one the most challenging examples of 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 cyclicly activated at fixed intervals (*periodic tasks*) ranging from a few milliseconds up to 100 ms, and some triggered at predefined rotation angles of the crankshaft (*angular tasks*) [12]. 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 [9]. 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 model for describing an AVR task was first proposed by Kim, Lakshmanan, and Rajkumar [13], 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. [14] derived a sufficient feasibility test for fixed priority scheduling, but assuming a constant engine speed. Davis et al. [10] 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. [5] 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 [6].

Other works addressed the analysis of AVR tasks under the Earliest Deadline First (EDF) scheduling algorithm. Guo and Baruah [11] proposed a speedup factor analysis and sufficient schedulability tests for AVR tasks scheduled with EDF. Biondi et al. [4] proposed a precise workload characterization generated by an AVR tasks that is used to derive a feasibility analysis under EDF scheduling.

The implementation of an AVR task also requires determining the precise engine speeds at which mode changes should occur. This problem has been addressed under EDF scheduling by Buttazzo, Bini, and Buttle [8], 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 in 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.

In our framework, the mode change issue is formulated as a design optimization problem, consisting in finding the switching speeds that optimize the overall system performance while guaranteeing the schedulability of the task set. We assume the knowledge of the performance function associated to each control implementation (they can be derived by fitting the

978-1-5090-1772-0/16/$31.00 ©2016 IEEE

test bench performance data to a family of relatively simple analytical functions). The consideration of such performance functions in the design optimization process together with the schedulability constraints is the main contribution of this work.

This paper 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 optimum 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 a finite granularity.

Paper structure. The remainder of the paper is structured as follows: Section II presents the model and the notation used throughout the paper. Section III formally states the problem considered in the paper. Section V presents the two heuristic approaches. Section VI describes the branch and bound algorithm. Section VII illustrates a set of experimental results aimed at comparing the proposed approaches. Finally, Section VIII states our conclusions and future work.

II. System model

This paper 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 inter-arrival of an AVR task is inversely proportional to the engine speed ω and 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 speed is denoted as $\vec{\omega}$.

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,

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

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

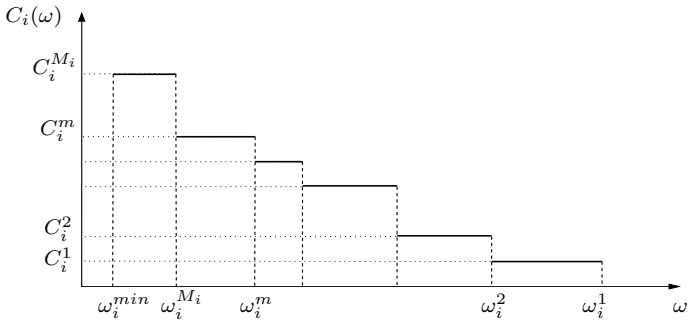

Figure 1: Computation time of an AVR task as a function of the speed at the job activation.

In the following, when a single AVR task is addressed, the task index is removed by the AVR task parameters for the sake of readability. To support the presentation of the derived results it is convenient to define the steady-state utilization of the j^{th} execution mode as a function of a generic 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 current switching speed ω^j.

A. Schedulability Analysis of AVR Tasks

In this paper, the schedulability analysis of mixed task sets consisting of periodic tasks and AVR tasks is performed using the test presented in [6]. 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 behaviour of AVR tasks causing mode changes as a function of the engine speed further complicates the identification of worst-case response times, preventing the treatment of such tasks as classical sporadic tasks. The analysis in [6] 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 [5], it is possible to limit the number of speed evolution patterns that have to be examined for deriving the worst-case interference. In [6] the response time computation has been 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.

III. Problem Definition

The analysis of this paper 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 only one 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 Λ^j are indexed such that their execution times are strictly increasing, that is, $C^j < C^{j+1}, \forall j = 1, \ldots, Q-1$.

Optimization variables. The objective of the proposed optimization algorithms is to find the set of switching speeds between modes of the AVR task and to assign a fixed priority to the AVR task and the periodic tasks to optimize the engine performance while guaranteeing the task set schedulability. Note that in the presence of AVR tasks the Rate-Monotonic priority assignment is not guaranteed to be optimal due to the large period variations of the AVR task [4]. The set of switching speeds is labeled as $\omega^j \in [\omega^{min}, \omega^{max}]$.

Constraints. The constraint we consider in our optimization 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 paper: 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*. The schedulability test considered in this paper is the one presented in [6] and summarized in Section II-A.

Performance Metric. The effectiveness of a design solution is evaluated by assigning each implementation a performance function $f_j(\omega)$ which is monotonically increasing with C^j, that is, $\forall \omega,\ f_j(\omega) > f_{j'}(\omega) \Leftrightarrow C^j > C^{j'}$. The rationale behind such a restriction is that a more complex control implementation only makes sense if it improves the performance. The overall system performance over the entire speeds range, considering all the possible execution modes is defined as

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

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 paper we consider two families of performance functions. In the first, 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 k^j and the overall system performance is

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

The second set of functions represent control implementations in which the performance depends on the control law that is implemented, but also on 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 function ($j = Q$, with possibly multiple injections during the cycle) has the best performance, which is kept unchanged across the speed range ($f_Q(\omega) = 1$, with $k^{Q,2} = 0$), and is considered as a baseline for the other modes. Simplified control functions 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).

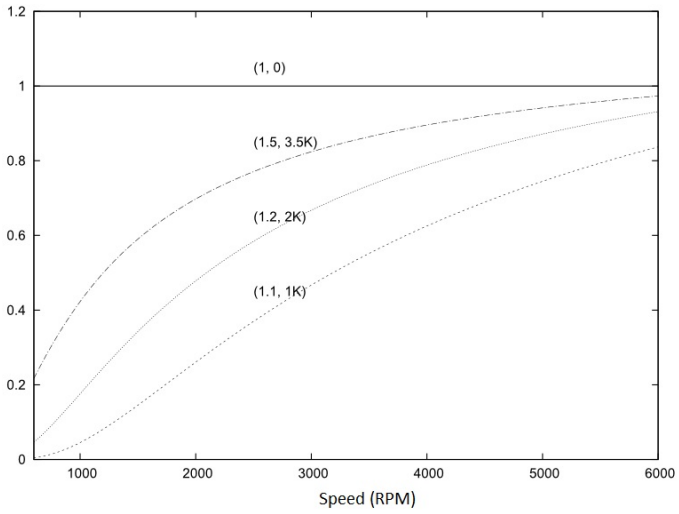

Figure 2: A set of exponential performance functions for selected values of $k^{j,1}, k^{j,2}$.

Formally, the first type of function is a special case of the

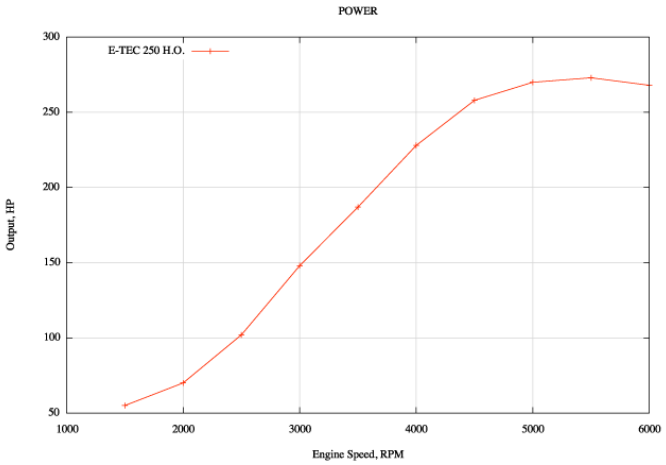

Figure 3: The power output curve of a the Evinrude E-TEC-250 engine (from continuouswave.com).

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. The integral in Equation (7) cannot be computed analytically, 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

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

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 1 (Optimization problem)**.**

$$\max \mathbb{P}(\omega^1, \ldots, \omega^Q)$$
$$\omega^1 > \omega^2 > \ldots > \omega^Q$$
$$\textit{such that} \text{ the system is schedulable.}$$

A. Running example

To illustrate the outcome of each algorithm and the optimization procedure, we use a running example (using a configuration from the EU INTERESTED project [1]) consisting of 4 periodic tasks with overall utilization $U = 0.825$, and an AVR task with 6 modes. The periods and the computation times of the periodic tasks are reported in Table I, whereas the computation times of the 6 modes of the AVR task are reported in Table II. To better explore the input space, AVR computation times 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 in μs).

	Task1	Task2	Task3	Task4
C	1000	6500	10000	10000
T=D	5000	20000	50000	100000

Table I: Parameters of the task set used 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

Table II: Computation times of the AVR task in the example.

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

IV. Reducing the design space

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 [3] 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 exists for the optimization problem for speeds $\omega^j \geq \omega^j_{(ub)}$. The determination of this upper bound also leads to an upper-bound $\mathbb{P}_{(ub)}$ on the maximum system performance.

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 implementation are active and (ii) the AVR task performs a direct mode change to the simplest control implementation having 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

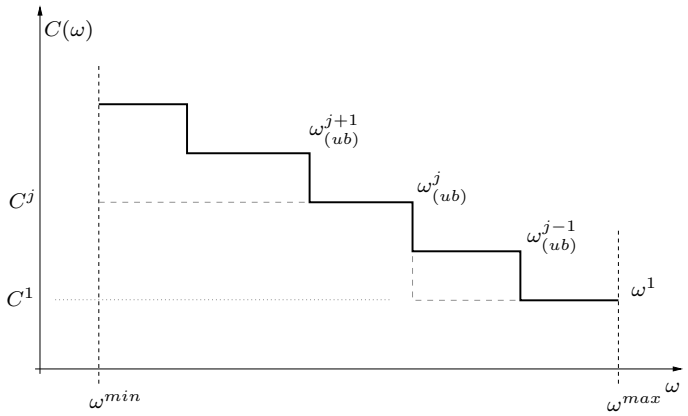

Figure 4: Computing the transition speed upper bounds.

condition, the algorithm computes the maximum speed $\omega^j_{(ub)}$ at which each control implementation Λ^j can be executed. In Figure 4, the (typically unfeasible) system modes configuration obtained by considering all the upper bounds transition speeds is shown as a thick line.

The approach is guaranteed to be correct because of the sustainability property of uniprocessor fixed-priority scheduling [7], which states that schedulability never improves when computation times are increased.

1: **procedure** COMPUTEUBS($C^1, \ldots, C^Q$)
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: $\omega^j_{(ub)} = \text{MAXBINSEARCH}(j, \mathcal{M})$;
6: **end for**
7: **return** $\{\omega^j_{(ub)}, j = 1, \ldots, Q\}$;
8: **end procedure**

Figure 5: Procedure for computing the upper-bounds for the switching speeds.

The algorithm to compute the switching speed upper bounds is shown in Figure 5. The first step determines whether there are modes that can be safely avoided. This is done by finding the implementation with the largest C^j that can safely be executed in ω^{min} and discarding those with higher index, and the implementation with the largest C that is feasible in ω^M 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 $M = Q$).

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). The maximum speeds are then determined for each mode $j > 1$ by a simple binary search (line 5) on the feasibility 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 $(\omega^j_{(ub)}, \omega^{max}]$.

A. Running example

The speed upper bounds for our running example are shown in Table III.

s	$\omega^1_{(ub)}$	$\omega^2_{(ub)}$	$\omega^3_{(ub)}$	$\omega^4_{(ub)}$	$\omega^5_{(ub)}$	$\omega^6_{(ub)}$	$\mathbb{P}_{(ub)}$
s=6	6500	6043	4848	3676	2996	1637	3504.84
s=8	6500	4285	3629	2996	1871	1214	2753.8

Table III: Upper bounds on the transition speeds for the considered example.

V. 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.

A. 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 $\delta \times p^j(\vec{\omega})$ (δ =50 in our experiments) 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 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 $\omega^1 < \omega^2 < \ldots < \omega^M$. 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 ω^1 for the first mode. This is often suboptimal (see Figure 6, for a space with only two speeds, with the suboptimal solution $\vec{\omega_{so}}$, the gray curve lines join the set of points with the same performance value, the black line the feasibility boundary).

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 $\Lambda^j, j > 2$.

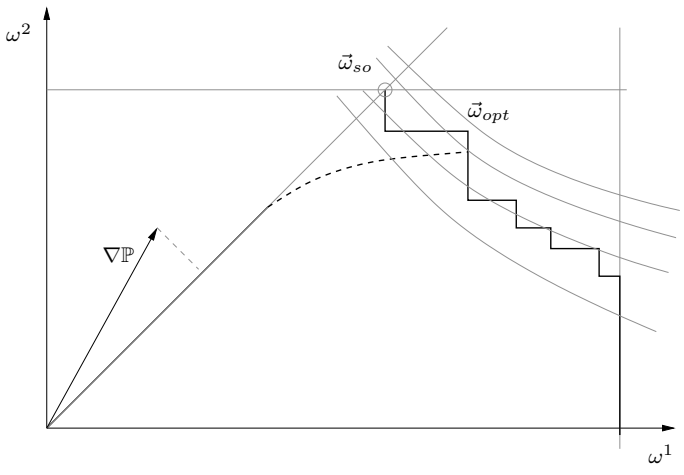

Figure 6: The modified gradient-based search.

To improve results, the gradient-driven 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:

$$z^j(\omega) = 1 - e^{\left(\frac{\omega^j_{(ub)} - \omega}{\omega^j_{(ub)}}\right)^2}. \quad (10)$$

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

$$T^j(\vec{\omega}) = z^j(\omega^j) + \frac{p^j(\omega^j)}{p^{max}}, \quad (11)$$

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 with a normalization step of $\frac{1}{p^{max}}$.

Each switching speed ω^j is progressively increased with rate $\delta \times T^j(\vec{\omega})$ (δ = 5rpm in our experiments), generating a sequence of speeds $\vec{\omega}(0), \vec{\omega}(1), \ldots, \vec{\omega}(k), \ldots$, in 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 IV. The effect of the corrected gradient trajectory is shown in Figure 6 (dashed line). From Equation (10), when approaching the upper-bound $\omega^j_{(ub)}$ for a mode j, the penalty term $z^j(\omega)$ is lowered 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 speed 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 having the maximum gradient coefficient, until no further steps can be performed without violating the schedulability constraint.

```
1:  procedure GradientSearch(C^1, ..., C^Q)
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
```

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

1) Running example: The application of the corrected gradient heuristic to our running example provides the results reported in Table IV.

s	ω^1	ω^2	ω^3	ω^4	ω^5	ω^6	$\mathbb{P}$	
6	6500	3817	3481	3146	2959	1598	3053.2	87.1% of $\mathbb{P}_{(ub)}$
8	6500	1460	1419	1366	1361	1168	1934.2	70.2% of $\mathbb{P}_{(ub)}$

Table IV: Results of the application of the corrected gradient heuristic to the running example.

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.

B. 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 (defined as $U_{(ub)} = U(\vec{\omega_{ub}})$) provides more freedom for the other speeds, that is, allows 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. Both considerations are combined in the Utilization and Performance-Driven backwards search.

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

$$\omega^j(k+1) = \omega^j(k) - \delta \times R^j(\vec{\omega}(k)), \qquad \forall j = 2, \ldots, Q \quad (12)$$

until a feasible set of speeds is found. The reduction step $R^j(\vec{\omega})$

is

$$R^j(\vec{\omega}) = max(\widehat{U}^j + \widehat{P}^j(\vec{\omega}), R_{min}). \quad (13)$$

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

$$\widehat{U}^j = \frac{U^j - U^{min}}{U^{max} - U^{min}}, \quad (14)$$

with

$$U^{max} = \max_j\{U^j\} \text{ and } U^{min} = \min_j\{U^j\}. \quad (15)$$

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

$$\widehat{P}^j(\vec{\omega}) = \frac{p^{max} - p^j(\vec{\omega})}{p^{max} - p^{min}}, \quad (16)$$

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

Figure 8 illustrates the pseudo code for the proposed utilization and performance-driven 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 V-A.

```
1:  procedure BACKWARDSSEARCH(C^1, ..., C^Q)
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
```

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

1) Running example: The application of the backwards search heuristic to our running example provides the results reported in Table V.

s	ω^1	ω^2	ω^3	ω^4	ω^5	ω^6	$\mathbb{P}$	
6	6500	6039	4836	3672	2899	1630	3480.2	99.3% of $\mathbb{P}_{(ub)}$
8	6500	4282	3194	2887	1868	1050	2644.0	96.0% of $\mathbb{P}_{(ub)}$

Table V: Results of the application of the backwards search heuristic to the running example.

The backwards search heuristic is very effective. The running example is not a special case, as shown in our experimental results. 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).

VI. 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 towards 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.

A. 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 $\delta = 15$ 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 only 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).

At each iteration with index j (j goes from 3 to Q), 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 $j = Q$), the speed ω_s^j is 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, of 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 increased up to Q, all the possible speed combinations (with granularity δ) are tried. The index j (controlling the speed that is selectively reduced) starts from 3 because it is always $\omega^1 = \omega_M$ and reducing ω^2 is pointless because ω^1 cannot be further increased. Also, it is necessary that ω_s^Q is the largest possible value that allows the execution of the other modes, because ω_s^Q is the only transition speed that is not increased in the search.

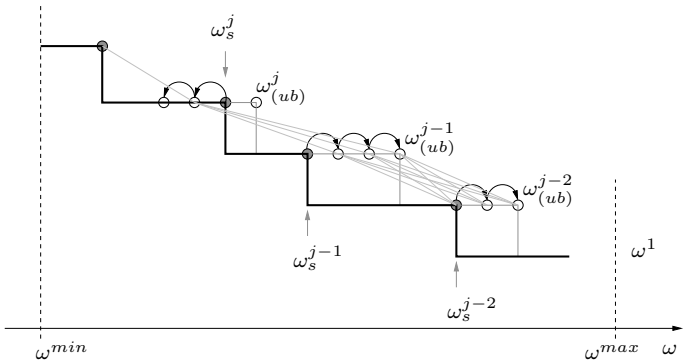

Figure 9: The optimization algorithm as a branch and bound search in the domain of the ω.

At any point in time, the search algorithm keeps track of the best performance solution found until then (initialized with the performance of the solution found by the backwards search heuristic).

At each iteration, the algorithm performs a pruning on the speed subtree when, after a reduction of $m_j \times \delta$ of ω_s^j, the current best performance value cannot be improved by any solution available in the subtree. The performance of all the solutions available in the current subtree are upper bound by the performance of the set of speeds $\{\omega_s^Q, \ldots, \omega_s^{j+1}, \omega_s^j - m_j \times \delta, \omega_{(ub)}^{j-1}, \ldots, \omega_{(ub)}^2, \omega^1\}$ constructed by leveraging on the knowledge of the speed upper bounds.

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). Next, 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.

The execution of the algorithm shows how the optimum performance often results in a configuration in which the largest speed decrease is for the mode with highest local utilization. This was the motivation for the deriving the utilization-driven backwards search heuristics.

The branch and bound computes solutions of very good quality at the expense of time. A set of experiments (see Section VII-C) 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.

B. 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 VI shows a summary of the results for the case with $s = 8$.

Algo	ω^1	ω^2	ω^3	ω^4	ω^5	ω^6	$\mathbb{P}$
H_∇	6500	1460	1419	1366	1361	1168	1934.2 70.2% of $\mathbb{P}_{(ub)}$
H_{BS}	6500	4282	3194	2887	1868	1050	2644.0 96.0% of $\mathbb{P}_{(ub)}$
BB	6500	4274	3556	2778	1858	1044	2665.9 96.8% $\mathbb{P}_{(ub)}$
UB	6500	4285	3629	2996	1871	1214	$\mathbb{P}_{(ub)} = 2753.8$

Table VI: Results for the running example with $s = 8$ applying all the algorithms presented in this paper.

The table shows the typical result found in our experiments. Not only the backwards heuristic is very close to the upper bound, but 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 heuristic than it is to the upper bound.

VII. Experimental Results

This section reports a set of experimental results aimed at evaluating and comparing the approaches presented in this paper. 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 [10], resulting in $\alpha^+ = -\alpha^- = 1.62\ 10^{-4}$ rev/msec^2.

A. 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 [12]. The execution times of the periodic tasks are generated by the UUnifast algorithm [2]. 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 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.

B. 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.

C. Constant performance function

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 IV. 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 150000 different configurations.

For each value of s the performances 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. The normalized performance values with respect to the upper-bound 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 $c^{min} = 100$, $c^{max} = 1000$, $c^{sep} = 100$ 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.

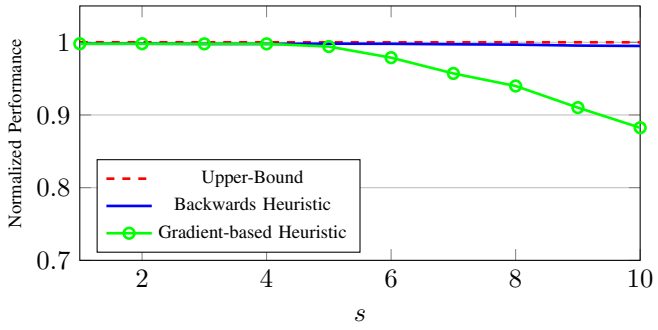

Figure 10: Performance of the heuristics normalized to the performance upper-bound as a function of s for $U^P = 0.5$.

Figure 10 shows the results for the case of a periodic task utilization $U^P = 0.5$. As shown by the graph, the backwards heuristic provides an extremely good performance, always greater 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. To save time in our experiments, we focused the remaining evaluation cases on the backwards search heuristic.

Figure 11 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 $s = 10$.

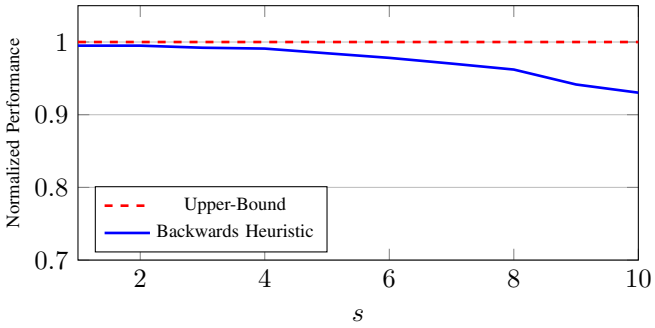

Figure 11: Performance of the heuristics normalized to the performance upper-bound as a function of s for $U^P = 0.75$.

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 run-time 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 12. 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.

The maximum observed run-time for the Backwards Search heuristic is 756 seconds with an average run-time of 5.6 seconds. For the Branch and Bound algorithm with precision δ =15 RPM we measured a maximum run-time of 16070 seconds with an average run-time of about 600 seconds. Such results have been obtained executing the algorithms on a machine equipped with Intel i7 processor running at 3.2 Ghz and 8Gb of RAM.

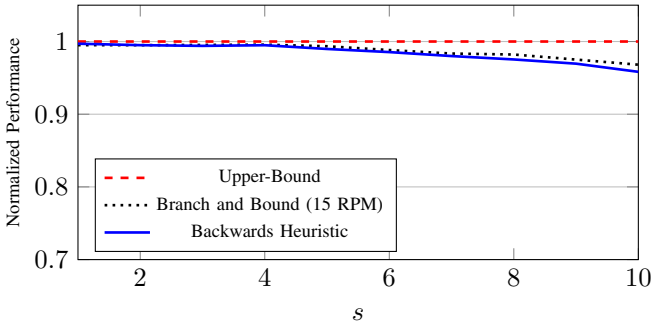

Figure 12: Performance of the heuristics normalized to the performance upper-bound as a function of s for $U^P = 0.75$.

D. Exponential performance functions

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

Figure 13 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 150000 different configurations.

As shown by the graph, the backwards search heuristic has a performance always greater than 99% of the upper-bound for $k^{2,max} = 50k^{2,min}$. In the case $k^{2,max} = 200k^{2,min}$ there is a slight degradation of the performance of the heuristic that reaches 96% of the upper-bound for $s = 10$.

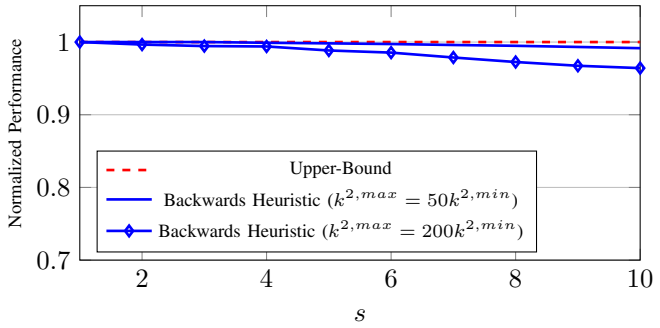

Figure 13: Performance of the heuristics normalized to the performance upper-bound as a function of s for $U^P = 0.75$ with exponential performance functions.

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 500000 configurations. The results are shown in Figure 14 and confirm the trend of Figure 13, showing a graceful and quite limited degradation for increasing values of the ratio $k^{2,max}/k^{2,min}$.

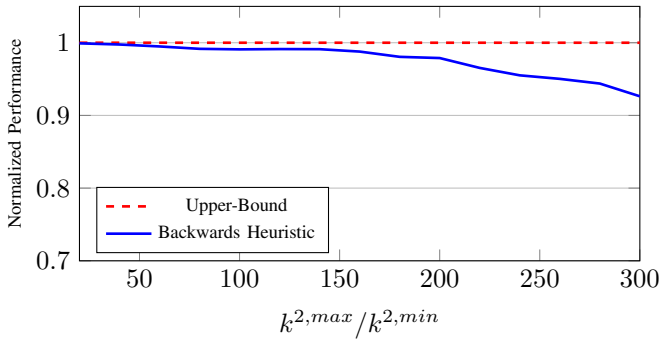

Figure 14: Performance of the heuristics normalized to the performance upper-bound as a function of $\frac{k^{2,max}}{k^{2,min}}$ for $U^P = 0.75$ and $s = 7$.

VIII. Conclusions

The problem of performance oriented design of transition speeds in a (fuel injection) system with adaptive variable rate tasks is discussed by presenting a set of optimization algorithms that apply to 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 extremely close to the actual optimum value and allow the computation of the optimum with finite resolution in many cases. Future work include the analysis of actual performance data and improving the schedulability constraint to allow for temporary overload conditions.

References

[1] INTERESTED, European project, Cordis project description. URL: http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/factsheets/interested.pdf.

[2] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), 2005.

[3] E. Bini, M. D. Natale, and G. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1-3):5–30, August 2008.

[4] A. Biondi, G. Buttazzo, and S. Simoncelli. Feasibility analysis of engine control tasks under EDF scheduling. In *Proc. of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10, 2015.

[5] A. Biondi, A. Melani, M. Marinoni, M. D. Natale, and G. Buttazzo. Exact interference of adaptive variable-rate tasks under fixed-priority scheduling. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 8-11, 2014.

[6] A. Biondi, M. D. Natale, and G. Buttazzo. Response-time analysis for real-time tasks in engine control applications. In *Proceedings of the 6th International Conference on Cyber-Physical Systems (ICCPS 2015)*, Seattle, Washington, USA, April 14-16, 2015.

[7] A. Burns and S. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[8] G. Buttazzo, E. Bini, and D. Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *Proc. of the Int. Conference on Design, Automation and Test in Europe*, Dresden, Germany, March 24-28, 2014.

[9] D. Buttle. Real-time in the prime-time. In *Keynote speech at the 24th Euromicro Conference on Real-Time Systems*, Pisa, Italy, July 12, 2012.

[10] R. I. Davis, T. Feld, V. Pollex, and F. Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *Proc. 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, Berlin, Germany, April 2014.

[11] Z. Guo and S. Baruah. Uniprocessor EDF scheduling of avr task systems. In *Proc. of the ACM/IEEE 6th International Conference on Cyber-Physical Systems (ICCPS 2015)*, Seattle, USA, April 2015.

[12] L. Guzzella and C. H. Onder. *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer-Verlag, 2010.

[13] J. Kim, K. Lakshmanan, and R. Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *Proc. of the Third IEEE/ACM Int. Conference on Cyber-Physical Systems (ICCPS 2012)*, pages 28–38, Beijing, China, April 2012.

[14] V. Pollex, T. Feld, F. Slomka, U. Margull, R. Mader, and G. Wirrer. Sufficient real-time analysis for an engine control unit with constant angular velocities. In *Proc. of the Design, Automation and Test Conference in Europe*, Grenoble, France, March 18-22, 2013.