Response-Time Analysis for Real-Time Tasks in Engine Control Applications

Alessandro Biondi
Scuola Superiore Sant'Anna
Pisa, Italy
alessandro.biondi@sssup.it

Marco Di Natale
Scuola Superiore Sant'Anna
Pisa, Italy
marco.dinatale@sssup.it

Giorgio Buttazzo
Scuola Superiore Sant'Anna
Pisa, Italy
giorgio.buttazzo@sssup.it

ABSTRACT

Engine control systems include computational activities that are triggered at predetermined angular values of the crankshaft, and therefore generate a workload that tends to increase with the engine speed. To cope with overload conditions, a common practice adopted by the automotive industry is to design such angular tasks with a set of modes that switch at given rotation speeds to adapt the computational demand. For this reason, these tasks are referred to as adaptive variable-rate (AVR). This paper presents an exact response time analysis for engine control applications consisting of periodic and AVR tasks scheduled by fixed priority. The proposed analysis is first presented for task sets with a single AVR task, and then extended to consider multiple AVR tasks related to a common rotation source. A number of experimental results are reported to validate the proposed approach and compare it against an existing sufficient test.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION BASED SYSTEMS**]: Real-time and embedded systems; D.4.7 [**Organization and Design**]: Real-time systems and embedded systems

1. INTRODUCTION

Engine control is a typical example of Cyber-Physical System (CPS), because the software that controls injection and combustion must be designed taking into account several physical characteristics of the engine and the combustion process taking place within it. The timing properties of the control software and its reactions define the performance of the engine with respect to fuel efficiency and pollutant generation, and must be tuned to avoid damaging the engine because of knocking [10].

One of the most peculiar features of engine control software is to require the execution of different kinds of computational activities: while some control tasks are activated by a timer at a fixed time intervals (*periodic tasks*), other control tasks are triggered at predefined rotation angles of the crankshaft (*angular tasks*), thus generating a dynamic workload that strictly depends on the engine speed: the higher the speed, the higher the activation rate. In a typical engine control application, periodic tasks have periods ranging from a few milliseconds up to 100 ms, while angular tasks are activated every single, half, and possibly quarter revolution of the crankshaft [10]. Tasks are normally scheduled using a fixed priority assignment, which cannot be optimal (in the sense of feasibility) due to the large period variations of angular tasks.

A major problem in engine control systems is that, at high engine speeds, the computational demand can generate an overload condition that, if not properly handled, could introduce large delays on control responses, with a significant performance degradation of one or more control functions [6].

A common solution often adopted in automotive applications to prevent such overload situations is to develop angular tasks that adapt their computational demand at different speeds, by changing their functionality [7]. This is typically done by defining a set of execution modes, each operating within a given speed range. For this reason, angular tasks are often referred to as *adaptive variable-rate* (AVR) tasks. In this work, AVR task is used as a synonym for angular task.

Unfortunately, the schedulability analysis of AVR tasks cannot be addressed using classical real-time methods. For instance, classical mode change analysis [13, 14, 17] cannot be applied to engine control systems, because the activation rate of an AVR task changes continuously. Likewise, the elastic task model proposed by Buttazzo et al. [3, 5] cannot be used to represent AVR tasks. In fact, in the elastic approach, an overload condition is not managed by the task itself by scaling its functions, but through a global resource manager, which reduces the utilization of the other tasks by enlarging their periods.

The schedulability analysis of real-time applications including both regular and AVR tasks has recently been addressed in the literature under different models and assumptions. One of the first models suitable for describing AVR tasks was proposed by Kim, Lakshmanan, and Rajkumar [11], denoted by the authors as rhythmic task model. The results presented in this work, however, were derived under very restrictive assumptions and apply to a single AVR task hav-

ICCPS '15 April 14 - 16, 2015, Seattle, WA, USA

ACM 978-1-4503-3455-6/15/04 ...$15.00.
http://dx.doi.org/10.1145/2735960.2735963

ing the highest priority and a variable period always smaller than the periods of the other tasks. In addition, relative deadlines are assumed to be equal to periods and priorities are assigned based on the Rate-Monotonic algorithm.

Later, Pollex et al. [12] derived a sufficient feasibility test for fixed priority scheduling, but under the assumption of a constant engine speed.

A more general schedulability analysis of AVR tasks under fixed-priorities has been presented by Davis et al. [8], who derived a sufficient test using an Integer Linear Programming (ILP) solver. Their method, however, is based on a quantization of the instantaneous engine speed, which introduces pessimism in the analysis.

An exact characterization of the interference produced by an AVR task under fixed priorities has been presented by Biondi et al. [2], who used a search approach in the speed domain, making use of dominant speeds to reduce complexity and avoid speed quantization.

Buttazzo, Bini, and Buttle [4] presented the design and the analysis of a mixed set of regular and angular tasks under Earliest Deadline First (EDF) scheduling. However, all angular tasks are assumed to be triggered by independent rotation sources. Although the assumption of independency simplifies the analysis, it introduces additional pessimism, considering situations that cannot actually occur when tasks are related to the same rotation source.

Contributions. This paper has three main novel contributions:

- An exact response-time analysis is presented for hybrid task sets consisting of regular periodic tasks and AVR tasks related to a common rotation source (the engine), under fixed-priority scheduling. To the best of our knowledge, this is the first exact response-time analysis in dynamic conditions (i.e., considering acceleration) for mixed periodic and AVR tasks having arbitrary fixed priorities.
- An algorithm is provided to efficiently compute the response-time of a periodic task subject to the interference of higher priority AVR tasks.
- Several simulation results are presented to validate the proposed approach and compare the performance of the exact schedulability test with the sufficient ILP-based test proposed in [8].

Paper structure. The remainder of the paper is structured as follows: Section 2 presents the model and the notation used throughout the paper. Section 3 briefly recalls how to compute the interference produced by a single AVR task. Section 4 presents the method for deriving the exact response time of each application task assuming the presence of a single AVR task, later extending the analysis to multiple AVR tasks related to a common rotation source and having the same angular period. Section 5 illustrates a set of experimental results aimed at validating the proposed approach. Finally, Section 6 states our conclusions and future work.

2. SYSTEM MODEL

This section introduces the notation adopted throughout the paper and the models used for the rotation source and the computational tasks. The rotation source considered in this paper 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 $[\alpha^-, \alpha^+]$. For the purpose of the analysis, the acceleration is assumed to have a negligible variation during a full revolution of the crankshaft.

2.1 Task Model

The task set considered in this work consists of n real-time preemptive tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task can be either a regular *periodic* task, or an AVR task, activated at specific crankshaft rotation angles. The subset of regular periodic tasks is denoted by Γ_P, whereas the subset of angular tasks is denoted by Γ_A, so that $\Gamma = \Gamma_P \cup \Gamma_A$ and $\Gamma_P \cap \Gamma_A = \emptyset$. Whenever needed, an AVR task may also be denoted as τ_i^*.

Both types of 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 angular 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:

$$\theta_i = \Phi_i + k\Theta_i, \quad \text{for } k = 0, 1, 2, \ldots$$

This means that the period 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.

In engine control, the angular phase Φ_i is 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 $\theta = 0$.

An AVR task τ_i^* is typically implemented as a set $\mathcal{M}_i$ of M_i execution modes, each consisting of a different functionality, valid in a predetermined range of rotation speeds. Mode m of an AVR task τ_i^* is characterized by a WCET C_i^m and is valid in a speed range $(\omega_i^{m+1}, \omega_i^m]$, 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 computation time of a generic AVR job $J_{i,k}$ can be expressed as a non-increasing step function $\mathcal{C}_i$ of the instantaneous speed ω at its release, that is,

$$C_{i,k} = \mathcal{C}(\omega) \in \{C_i^1, \ldots, C_i^{M_i}\}. \quad (2)$$

An example of $\mathcal{C}$ function is illustrated in Figure 1.

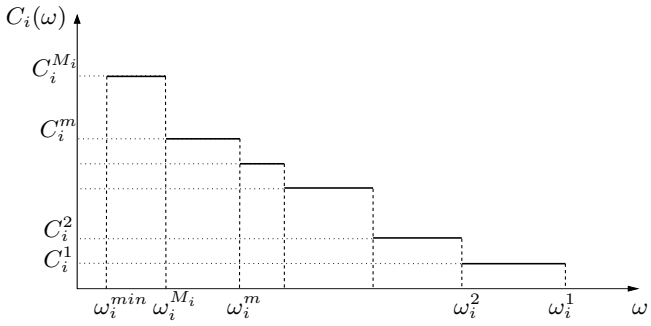

Figure 1: Computation time of an AVR task as a function of the speed at the job activation.

2.2 Rotation Source Model

For the purpose of analyzing the schedulability of engine control applications consisting of AVR tasks, it is crucial to characterize the relation between the task parameters and the dynamics of the engine. Following the approach proposed by Buttazzo et al. [4], if (ω_k, α_k) is the state of the engine at time t_k, the next job release time of an AVR task can be computed (assuming a constant acceleration α_k) as:

$$T_i(\omega_k, \alpha_k) = \frac{\sqrt{\omega_k^2 + 2\Theta_i\alpha_k} - \omega_k}{\alpha_k}. \quad (3)$$

The corresponding relative deadline in the time domain can be computed by considering the earliest value given by the maximum acceleration α^+, that is

$$D_i(\omega_k) = \frac{\sqrt{\omega_k^2 + 2\Delta_i\alpha^+} - \omega_k}{\alpha^+}. \quad (4)$$

In a similar way, the instantaneous rotation speed at the next job release can be computed (assuming constant acceleration during the angular period) as $\Omega(\omega_k, \alpha_k) = \omega_k + \alpha_k T_i(\omega_k, \alpha_k)$, which gives:

$$\Omega_i(\omega_k, \alpha_k) = \sqrt{\omega_k^2 + 2\Theta_i\alpha_k}. \quad (5)$$

If a job $J_{i,k}$ is released when the engine has an instantaneous speed ω_k, the interarrival time $\widetilde{T}_i(\omega_k, \omega_{k+1})$ to the next job $J_{i,k+1}$ released with instantaneous speed ω_{k+1} (if reachable with the acceleration bounds), can be obtained by Equation (3), substituting α_k from Equation (5), which gives:

$$\widetilde{T}_i(\omega_k, \omega_{k+1}) = \frac{2\Theta_i}{\omega_k + \omega_{k+1}}. \quad (6)$$

Similarly, given a job $J_{i,k}$ released at instantaneous speed ω_k, the minimum interarrival time $\widetilde{T}_i^m(\omega_k)$ to have the next job $J_{i,k+1}$ released in mode m can be computed as

$$\widetilde{T}_i^m(\omega_k) = \widetilde{T}_i(\omega_k, \omega^m) = \frac{2\Theta_i}{\omega_k + \omega^m}. \quad (7)$$

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 help the reader in following the adopted notation, Table 1 summarizes the main symbols used throughout the paper.

Symbol	Description
τ_i	i^{th} periodic task
τ_i^*	i^{th} AVR task
$\mathcal{C}_i(\omega)$	WCET of τ_i^* as a function of the instantaneous speed
C_i^m	WCET of mode m of τ_i^*
ω_i^m	Maximum speed for mode m of τ_i^*
Θ_i	Angular period of τ_i^*
$T_i(\omega_k, \alpha_k)$	Inter-arrival time between the k^{th} and $(k+1)^{\text{th}}$ job instances, assuming the acceleration is α_k
$D_i(\omega_k)$	Relative (temporal) deadline for a job released at speed ω_k
$\Omega_i(\omega_k, \alpha_k)$	Engine speed at the release of job $J_{i,k+1}$ assuming that job $J_{i,k}$ is released with speed ω_k and acceleration α_k
$\widetilde{T}_i(\omega_k, \omega_{k+1})$	Inter-arrival time between a job released at speed ω_k and the following at speed ω_{k+1}
$\widetilde{T}_i^m(\omega_k)$	Minimum inter-arrival time between a job released at speed ω_k and the following in mode m
$\mathcal{S}(t)$	Set of job sequences for an AVR task in a time window $[0, t]$
$\mathcal{CS}(t)$	Set of *critical* job sequences for an AVR task in a time window $[0, t]$
$I^{(s)}(t)$	Interference of a job sequence s of an AVR task
$I(t)$	Interference envelope of an AVR task
$R_i^{(s)}$	Response time of task τ_i interfered by the job sequence s of an AVR task
R_i	Response time of task τ_i
$\overline{R_i}$	Upper-bound for the response time of task τ_i

Table 1: Main notation used throughout this paper.

3. INTERFERENCE COMPUTATION AS A SEARCH PROBLEM

This section briefly recaps the approach proposed in [2] and introduces the concepts and the notation used in the response time analysis presented in the next section.

Let us consider a job J_0 of an AVR task τ^* released at time $t = 0$ with speed ω_0, as illustrated in Figure 2, and suppose that the job executes for its WCET $\mathcal{C}(\omega_0)$. Since the engine is subject to an acceleration $\alpha \in [\alpha^-, \alpha^+]$, there are *infinite* possible time instants for the next job release. The earliest possible job release corresponds to the maximum acceleration α^+ and would occur after $T(\omega_0, \alpha^+)$ time units, whereas the latest job release corresponds to the maximum deceleration α^-, occurring after $T(\omega_0, \alpha^-)$ time units.

Depending on the instantaneous engine speed at its activation, the next job J_1, can execute in different modes with different WCETs. The function $i_{\omega_0}(t)$ reported in Figure 2 is denoted as *single-job interference* and represents the envelope of the interference contribution among all the possible subsequent jobs. The instantaneous speed ω_1 at the activation of J_1 is constrained in the range $[\Omega(\omega_0, \alpha^-), \Omega(\omega_0, \alpha^+)]$

and its value depends on the engine acceleration during the inter-arrival time.

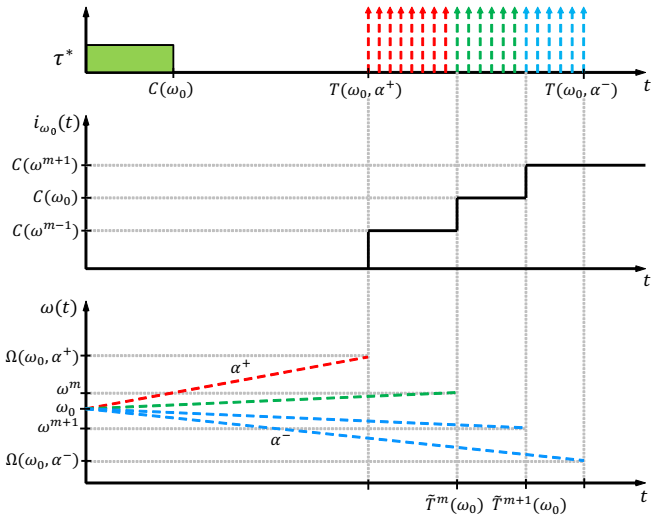

Figure 2: Possible activations after a job released at speed ω_0.

The same reasoning applies to each job generated after J_1, leading to a *tree* of possible job sequences as illustrated in Figure 3.

The computation of the interference of an AVR task can be seen as a search problem in the speed domain, analyzing all possible job sequences in a given time window and the composition of the corresponding single-job interferences. Considering that the speed domain is a continuum, the search tree is infinite (i.e., it contains an infinite set of job sequences). This means that, if a brute-force approach is applied, a quantization on the speed domain is required to achieve a solution in a finite time.

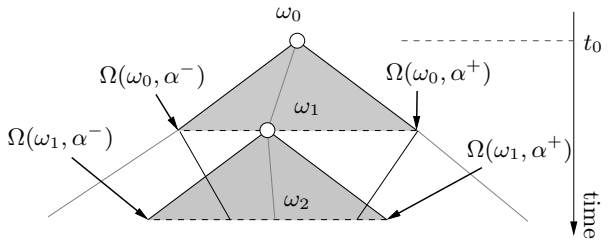

Figure 3: Search tree representing the possible job sequences for an AVR task.

Figure 4 shows the pseudo code for a brute-force search of the tree using quantization. Assuming to start from a job J_0 released at speed ω_0 at $t = 0$, the procedure is called as `Interference`$(\omega_0, \mathcal{C}(\omega_0), 0)$. The MaxTime parameter represents the maximum time interval in which the interference has to be computed (further details on this approach can be found in [2]). Each recursive instance of the `Interference` procedure represents a job activated at time t with instantaneous speed ω and Π is the sum of all the computational request imposed by the previous jobs. The procedure

```
1: procedure Interference(ω, Π, t)
2:     if t > MaxTime then return ;
3:     end if
4:     UpdateInterference(Π, t);
5:     for α = α⁻ to α⁺ step Δα do
6:         ω^next ← Ω(ω, α);
7:         T^next ← T(ω, α);
8:         Π^next ← Π + C(ω^next);
9:         Interference(ω^next, Π^next, t + T^next);
10:    end for
11: end procedure
```

Figure 4: Procedure for computing the interference of an AVR using brute-force on the search domain.

UpdateInterference keeps track of the computational request accumulated at time t; then the algorithm prepares for the next recursive call exploring the speed domain with quantization. Besides providing only an approximate (and possibly unsafe) analysis, this approach is very expensive in terms of computational complexity and intractable for most practical cases.

For a system consisting of a mixed set of periodic tasks and AVR tasks (having the same angular phase and activated by a single rotation source), the critical instant of a task (i.e., the activation time that produces its longer response time) occurs when it is activated at the same time of all higher priority tasks. Without loss of generality, we assume that the critical instant occurs at time $t = 0$. In addition, the critical instant must be considered for each instantaneous speed ω_0 of the AVR set, hence, the search algorithm has to be repeated for each possible speed $\omega_0 \in [\omega^{min}, \omega^{max}]$ requiring again a quantization and further increasing the computational complexity.

Definition 1. A job sequence s for an AVR task τ^* is composed of a set of consecutive released jobs $J_0, \ldots, J_{n_s}$, and a set of speeds $\omega_0, \ldots, \omega_{n_s}$, where each element $\omega_k, k = 0, \ldots, n_s$ represents the speed at the activation of job J_k.

The interference of an AVR task is characterized by an infinite set of possible job sequences in a given time window $[0, t]$. Let $\mathcal{S}(t)$ be such a set. Intuitively, each path in the search tree represents a job sequence.

Each sequence $s \in \mathcal{S}(t)$ generates an interference $I^{(s)}(t)$, which is a function of ω_0 and $\omega_k, k = 1, \ldots, n_s$, because it depends on the speed evolution pattern experienced by the AVR task. In general $I^{(s)}(t)$ can be expressed as

$$I^{(s)}(t) = \mathcal{C}(\omega_0) + \sum_{k=1}^{n_s} \mathcal{C}(\omega_k)\ \mathsf{step}\left(t - \sum_{j=1}^{k} \widetilde{T}(\omega_{j-1}, \omega_j)\right),$$

where

$$\mathsf{step}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}.$$

In addition, it is convenient to define the partial contribution $I^{(s)}(t, i, \omega)$ to $I^{(s)}(t)$ starting from the intermediate nodes (corresponding to job activations) of the job tree. $I^{(s)}(t, i, \omega)$

is defined as the worst-case contribution to the interference from all the jobs of index i and higher, assuming the activation of the i^{th} job occurs when the speed is ω.

Ideally, to cope with all possible speed evolution patterns, all the job sequences $s \in \mathcal{S}(t)$ have to be considered to obtain a characterization of the interference. However, Biondi et al. [2], defined the conditions for finding a set of *dominant speeds* and a set of pruning rules to reduce the infinite speed domain to a discrete and limited set of critical job sequences. Similarly, the search tree for all possible ω_0 values can be reduced to a finite set of dominant initial speeds. Note that this approach allows computing the exact interference with a contained complexity and avoiding quantization!

Definition 2. Given a job instance J_i of an AVR task τ^*, the set of its *dominant speeds* $\mathcal{W} = \{\omega^d_{i,0}, \omega^d_{i,1}, \ldots\}$ is the set of engine speeds at the release time of J_i such that for any other engine speed $\omega \notin \mathcal{W}$ there exists at least one $\omega^d_{i,j} \in \mathcal{W}$ such that $I^{(s)}(t, i, \omega^d_{i,j}) \geq I^{(s)}(t, i, \omega)$.

Dominant speeds can be computed by leveraging the conditions stated in Theorem 2 in [2]. Using the concept of dominant speeds, it is possible to define a *critical* job sequence for an AVR task.

Definition 3. A *critical* job sequence for an AVR task τ^* is a job sequence where each job is released at a dominant speed.

The set $\mathcal{CS}(t)$ of the possible critical job sequences in the time window $[0, t]$ can be computed through a pruned visit of the search tree in the speed domain (considering the possible dominant initial speeds and the following dominants at each following job).

The main property of the pruning-based methodology is that for each non-critical job sequence s' there exists a critical job sequence s whose interference dominates the one of s'. Formally, when a time window $[0, D]$ is considered for the interference characterization, we have that

$$\forall s' \notin \mathcal{CS}(D)\ \exists s \in \mathcal{CS}(D) \mid \forall t \in [0, D]\ I^{(s)}(t) \geq I^{(s')}(t). \quad (8)$$

Based on this result, the worst-case interference caused by an AVR task τ^* will result from a sequence of jobs belonging to one of the critical sequences.

4. EXACT RESPONSE-TIME ANALYSIS

In this section we derive a response time analysis for a mixed set of tasks having constrained deadline $D_i \leq T_i$. We first consider the case in which a single AVR task interferes on a periodic task set, and then the opposite case in which a periodic task set creates interference on a single AVR task. The extension to multiple AVR tasks activated by the same rotating source is addressed in Section 4.1.2 and Section 4.3.

4.1 Response-time of a Periodic Task Interfered by an AVR Task

We first derive the response time of a periodic task τ_i interfered by a set of regular periodic tasks and a single AVR task τ^*, all having higher priority than τ_i. In the following, $hp(i)$ denotes the set of periodic tasks having higher priority than τ_i. To simplify the notation, the set $\mathcal{CS}$ is used as a shortcut for $\mathcal{CS}(D_i)$, denoting the set of critical job sequences in the interval $[0, D_i]$.

Given a particular job sequence s of τ^*, the response time of τ_i denoted as $R_i^{(s)}$, can be expressed as

$$R_i^{(s)} = \min \left\{ t \mid C_i + I^{(s)}(t) + \sum_{j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j = t \right\}.$$

The challenging part in the analysis is clearly the computation of the interference $I^{(s)}(t)$ imposed by the AVR task. Exploiting the pruning methodology presented in [2], the following theorem demonstrates that the response time of τ_i can be computed only considering the critical job sequences in the interval $[0, D_i]$.

Theorem 1. *The response-time R_i of a periodic task τ_i interfered by an AVR task τ^* is the maximum response-time over all possible critical job sequences generated by τ^*, that is*

$$R_i = \max_{s \in \mathcal{CS}} R_i^{(s)}. \quad (9)$$

Proof. Each critical job sequence $s \in \mathcal{CS}$ leads to an idle time after time $R_i^{(s)}$. By contradiction, suppose that there exists a non-critical job sequence $s' \notin \mathcal{CS}$ such that $R_i^{(s')} > R_i$; then, to avoid idle-time before time R_i, it must be that $I^{(s')}(R_i^{(s)}) > I^{(s)}(R_i^{(s)})\ \forall s \in \mathcal{CS}$. Hence, for each critical job sequence s, we have a time instant at which the interference $I^{(s)}$ is dominated by the one of the non-critical job sequence s'. This contradicts the main property of critical job sequences expressed by Equation (8), hence the theorem follows. □

In [2], the maximum over all the possible sequences s (leading to further branching in the computation of the response time) was avoided by using the *interference envelope* $I(t)$ for τ^*, defined as

$$I(t) = \max_{s \in \mathcal{S}(t)} I^{(s)}(t).$$

and computing a response time upper bound $\overline{R}_i$ as

$$\overline{R}_i = \min \left\{ t \mid C_i + I(t) + \sum_{j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j = t \right\}.$$

However, as highlighted by Stigge and Yi [15,16] in the context of the digraph real-time task model, this approach introduces a pessimism in the analysis, preventing the derivation of the exact response time. In fact, the speed sequences that generate the critical job sequences $s \in \mathcal{CS}$ (the functions contributing to the definition of $I(t)$) may be mutually exclusive. Thus, simply computing the maximum may lead to a sequence of speeds that cannot occur in practice. In other words, by computing the interference envelope we lose the information about the sequence of speeds that may generate the envelope.

To better clarify this point, consider we want to compute the response time of a period task τ_i executing with an AVR task τ^* having higher priority. Also, assume that the set

$\mathcal{CS}(D_i)$ is composed of only two sequences, i.e., $\mathcal{CS}(D_i) = \{s_a, s_b\}$. Figure 5 shows the two interferences $I^{(s_a)}(t)$ and $I^{(s_b)}(t)$ originated by the two job sequences s_a and s_b, respectively, and the corresponding interference envelope $I(t)$, computed as the maximum between $I^{(s_a)}(t)$ and $I^{(s_b)}(t)$. As clearly visible from the plots, the envelope $I(t)$ leads to a response time $\overline{R}$ much higher than those resulting from the two concrete interferences (note that in the graph the response time is the time instant at which the computational demand matches the processor supply). In other words, any concrete job sequence of τ^* contributing to $I(t)$ generates an idle-time before $\overline{R}$.

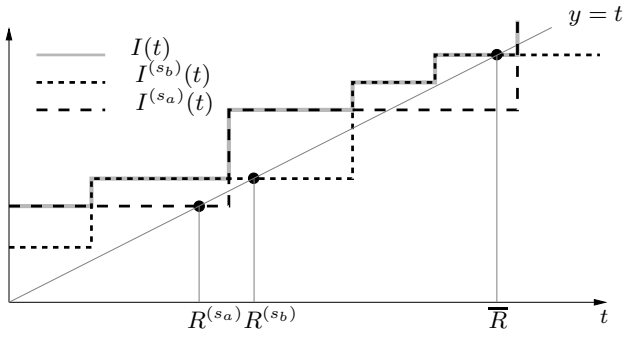

Figure 5: Example in which the interference envelope leads to an over-estimated response-time.

In general, since the determination of the time at which the processor is idle cannot be computed without a full knowledge of the higher priority task set, it is not possible to compute a priori the interference of an AVR task. In other words, the maximum response-time R_i is originated from different sequences $s \in \mathcal{CS}$ depending on the interference of the higher priority tasks. Hence, contrary to classical periodic tasks, to characterize the exact response-time it is not possible to abstract the interference of τ^* by using a single value of interference for each time instant t. The solution proposed in this paper consists of computing the interference *on the fly* while the response time of τ_i is computed, exploring the domain of the set $\mathcal{CS}$.

4.1.1 Algorithm for Computing the Response Time

According to Equation (9), to compute the response-time R_i, it is necessary to identify all possible critical job sequences $s \in \mathcal{CS}$ and then compute the response-time $R_i^{(s)}$ for each sequence s.

In this section, we present an algorithm to efficiently compute R_i by evaluating the critical job sequences on-the-fly and only when needed, providing additional pruning in the search of the speed space and significant speedup for the analysis.

The proposed algorithm (reported in Figure 7) performs a visit of the speed tree with pruning, using the concept of dominant speeds while at the same time discarding the job sequences that would result in an idle time earlier than one of the candidate response times. The algorithm operates recursively for increasing time values. At any point in time, the main function of the algorithm, `ResponseTime`, computes the contribution to the interference of one additional job activation. `ResponseTime` is called by passing the priority index i of the task for which the response time is computed (used to evaluate the contribution to the interference from the set of periodic tasks), the current speed ω (at the time the job is activated), the current time t, and the execution time requests Π (the contribution to the interference) accumulated up to time t. Note that the algorithm always terminates when current time t exceeds MaxTime, which is the maximum time (equal to the task deadline) for the search.

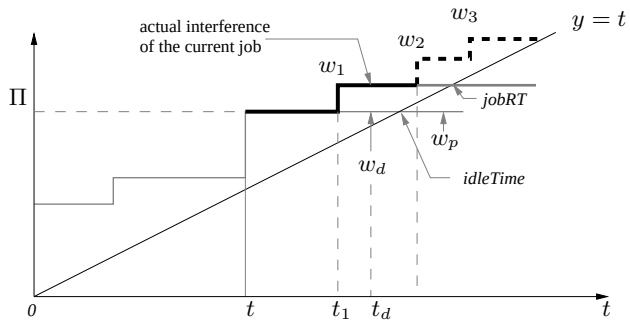

Figure 6: Pruning and branching in the algorithm for computing the response time.

Each time `ResponseTime` is called, ω is one of the dominant speeds and t the candidate point in time for the job activation. Given t and ω, the algorithm computes the next possible idle time (an example is illustrated in Figure 6). Next, it computes the times and speeds for the possible activations of the next job that dominate its execution time requests. The requests that occur before the next idle time (such as ω_1 in the figure) are considered by the algorithm, while those occurring after the idle time (such as ω_2 and ω_3 in the figure) are discarded. Among all the possible job activations that occur before the idle time, the algorithm selects the one that results in the candidate response time at the current step (in `singleJobRT`). The candidate response time is stored and, if it is still smaller than the deadline, the algorithm prepares for the next recursive call. At this point, the procedure getDominants(ω) computes the dominant speeds in the range of possible instantaneous speed at the activation of the next job, that is $[\Omega(\omega, \alpha^-), \Omega(\omega, \alpha^+)]$. Among those dominant speeds, there are not only the ones that contribute to the steps in the time request function for the current job (such as ω_1), but also additional ones (dominating the requests for a subset of the following jobs). For these dominants, the same pruning rule based on the idle time applies. If they result in activations before the idle time (such as ω_d in the figure), they are considered for the next recursive call, otherwise (ω_p in the figure), they are pruned.

Figure 8 shows the algorithm to check the schedulability of τ_i using the procedure `ResponseTime`. Procedure `SchedulabilityTest` starts by computing the initial dominant speeds in the full range allowed for engine speeds, (i.e., $[\omega^{min}, \omega^{max}]$). Then, for each initial dominant speed ω_0 we compute the response-time candidates: since all the candidates represent response-time values related to possible job sequences starting from ω_0, the maximum R of such candidates is taken as response time for speed ω_0. If the response-time R exceeds

the deadline D_i, then τ_i is not schedulable; otherwise, if R results lower (or equal) than D_i for each initial dominant speeds, then τ_i is schedulable.

```
1: procedure ResponseTime(i, ω, Π, t)
2:     idleTime ← getIdleTime(i, ω, Π, t);
3:     jobRT ← singleJobRT(i, ω, Π, idleTime);
4:     RTCandidates.add(jobRT);
5:
6:     if jobRT> MaxTime then return ;
7:     end if
8:
9:     dominants ← getDominants(Ω(ω, α⁻), Ω(ω, α⁺));
10:    for ω^next in dominants do
11:        T^next ← T̄(ω, ω^next);
12:        if t + T^next < idleTime then
13:            Π^next ← Π + C(ω^next);
14:            ResponseTime(i, ω^next, Π^next, t + T^next);
15:        end if
16:    end for
17: end procedure
```

Figure 7: Procedure for computing the response-time of a task τ_i.

```
1: procedure SchedulabilityTest(i)
2:     dominants ← getDominants(ω^min, ω^max);
3:     MaxTime ← D_i;
4:     for ω_0 in dominants do
5:         RTCandidates ← [ ];
6:         ResponseTime(i, ω_0, 0, 0);
7:         R ← Max(RTCandidates);
8:
9:         if R > D_i then
10:            return UNSCHEDULABLE;
11:        end if
12:    end for
13:    return SCHEDULABLE;
14: end procedure
```

Figure 8: Procedure describing the schedulability test for a task τ_i.

Surprisingly, the additional pruning in the search allowed by the idle-time detection makes the computation of the exact response-time more efficient than the computation of the over estimated response-time through the interference envelope $I(t)$.

4.1.2 Interference from Multiple AVR Tasks

In this section we extend the analysis by considering the possible interference from multiple AVR tasks with the same rotating source of activation events (the case is significant when considering engine-control application).

Since the set of jobs from the AVR tasks are activated exactly at the same time instant, the definition of *c*ritical instant does not change. The intereference of the tasks in the AVR (higher priority) set can be characterized as follows.

When computing the response time of a periodic task τ_i interfered by a set of AVR tasks $hp^A(\tau_i) = \{\tau_0^*, \tau_1^*, \ldots \tau_p^*\}$ having the same angular period and phase (assumed as 0 for convenience), the interference from the tasks in the AVR set is equivalent to the interference from a single task τ_k^* constructed as follows.

Consider the union of the mode speeds of the AVR tasks and sort them from ω^{max} to ω^{min}. The cardinality of the set gives the number of modes for τ_k^* (at most the sum of the number of modes for all the AVR tasks). Each mode m of τ_k^* is defined by the corresponding speed range $(\omega_k^{m+1}, \omega_k^m]$ and a worst case execution time

$$\mathcal{C}_k^m(\omega) = \sum_{j \in hp^A(\tau_i)} \mathcal{C}_j(\omega_k^m).$$

At this point, the approach presented in the previous section can be applied.

4.2 Response-Time of an AVR Task Interfered by Periodic Tasks

Let us now consider the response-time of an AVR task τ^* interfered by periodic tasks (assuming there are periodic tasks having higher priority than τ^*).

Since the response time of τ^* depends on the instantaneous speed ω_0 at which it is released, we have

$$R(\omega_0) = \min \left\{ t \mid \mathcal{C}(\omega_0) + \sum_{i \in hp(\tau^*)} \left\lceil \frac{t}{T_i} \right\rceil C_i = t \right\},$$

where $hp(\tau^*)$ denotes the set of periodic tasks having higher priority than τ^*.

Now, the dependency on the speed ω_0 can be removed by considering each mode of τ^*, so obtaining a response-time value for each mode, that is

$$R^m = \min \left\{ t \mid C^m + \sum_{i \in hp(\tau^*)} \left\lceil \frac{t}{T_i} \right\rceil C_i = t \right\},$$

with $m = 1, \ldots, M$.

Finally, to check the schedulability of τ^* we have to verify that $R^m \leq D(\omega^m)$ for each mode $m = 1, \ldots, M$.

4.3 Response Time of an AVR Task Interfered by other AVR Tasks

We now address the schedulability of an AVR task τ_i^* interfered by both periodic tasks and multiple AVR tasks having the same angular period of τ_i^*. When multiple AVR tasks with the same angular period are considered, only one job for each high-priority AVR task can produce interference, i.e., a single computation time must be accounted for. We denote $hp^A(\tau_i^*)$ as the set of AVR tasks having higher priority than τ_i^*.

As done in Section 4.2, the dependency from the speed $\omega_0 \in [\omega^{min}, \omega^{max}]$ can be removed by considering each mode m of τ_i^* and computing the reponse time for each m. Once the mode of τ_i^* is selected, each higher priority AVR task $\tau_j^*, j \in hp^A(\tau_i^*)$ may be in a finite set of modes $m_{j,k}, \ldots, m_{j,n}$ such that the intersection of the speed ranges for m and any of the $m_{j,p}$ with $k \leq p \leq n$ is not empty (as shown in Figure 9). The possible interference of each higher priority

AVR τ_j^* only changes at the boundary speeds of its modes. Hence, these are the only (finite) number of speeds that need to be considered.

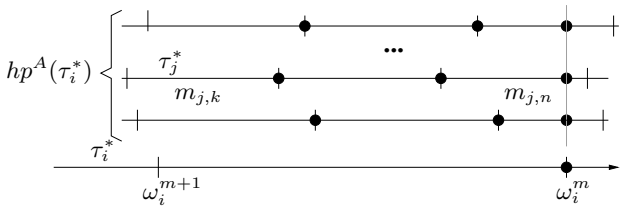

Figure 9: Identifying the contributions of the higher priority AVR task modes to τ_i^*.

Formally, for each mode m of τ_i^*, the following conditions must be satisfied:

$$\forall j \in hp^A(\tau_i^*) \cup \{i\}$$
$$\forall \omega_j^{m'} \in (\omega_i^{m+1}, \omega_i^m] \quad R_i^m(\omega_j^{m'}) \leq D_i(\omega_j^{m'}).$$

where

$$R_i^m(\omega) = \min \left\{ t \mid C_i^m + \sum_{j \in hp^A(\tau_i^*)} \mathcal{C}_j(\omega) \; + \sum_{j \in hp(\tau_i^*)} \left\lceil \frac{t}{T_j} \right\rceil C_j = t \right\}.$$

5. EXPERIMENTAL RESULTS

This section presents a set of experimental results aimed at comparing the exact schedulability analysis presented in this paper with the sufficient ILP-based analysis for AVR tasks presented by Davis et al. in [8]. Both schedulability tests have been implemented and applied to synthetic workloads for comparison. The ILP-based formulation has been implemented using the IBM CPLEX solver, whereas the proposed algorithm has been implemented as a MATLAB script. Tests have been executed on a desktop PC with processor Intel i7 running at 3.5 GHz.

Since the ILP-based analysis requires a quantization on the speed domain, a step of 100 RPM was adopted, as suggested by the authors. Our analysis discriminates 1 RPM in the computation of the dominant speeds. It is also worth noting that the approach presented in [8] considers a slightly different task model in which mode-change is triggered as a function of an estimation of the instantaneous speed through the average speed in the previous inter-arrival time. The ILP formulation [8] leads to some inconsistencies in the computation of the interference for low speed values: the problem has been fixed by the authors in a technical report [9], taken as a reference for our comparison.

We assume a rotation source ranging from $\omega^{min} = 500$ RPM to $\omega^{max} = 6500$ RPM as typical values for a production car engine. Similarly, values of acceleration have been selected [8] such that the engine is able to reach the maximum speed starting from the minimum one in 35 revolutions, obtaining $\alpha^+ = -\alpha^- = 1.62\ 10^{-4}$ rev/msec^2.

5.1 Workload Generation

In our experiments we considered a task set composed of n periodic tasks and an AVR task. Given an overall target utilization U^P for the set of periodic tasks, each periodic tasks is generated as follows:

- The utilization U_i of each task τ_i is randomly generated using the UUniFast [1] algorithm such that $\sum_{i=1}^{n} U_i = U^P$. The minimum utilization of each periodic task is $U^{min} = 0.005$;
- Task periods T_i are randomly generated (with a uniform distribution) in the range $[3, 100]$ msec;
- We considered implicit deadlines for each periodic task, i.e., $D_i = T_i$;
- Computation times are computed as $C_i = U_i T_i$.

Observe that the case of multiple AVR tasks with a common activation source can be modeled as a single AVR task (also called *representative AVR*) in which the definition of the modes, the speed boundaries for the modes, and the execution time for each mode are defined as a combination of the modes, boundary speeds and execution times of the tasks in the AVR set. For the purpose of our experiments, we simply represent the single AVR task resulting from the composition of the set.

Given a target utilization U^* for the representative AVR task, its parameters are generated as follows:

- $\Theta = 2\pi$ is the angular period (i.e., a task activation for each engine revolution);
- $\Delta = \Theta$ (implicit angular deadlines);
- The number of modes M has been randomly generated in the range $[M^{min}, M^{max}]$. The values defining the range are parameters for the definition of the experimental set;
- A random mode m' is selected to have the maximum utilization $U^{m'} = U^*$. The utilization U^m of the other modes $m \neq m'$ are randomly generated in the range $[0.85U^*, U^*]$, where U^* is an additional parameter;
- The maximum speed ω^m of each mode $m < M$ is randomly generated in the range $[1000, 6000]$ RPM. The maximum speed for mode 1 is always set at the maximum speed ω^{max}. Once the boundary speeds for the mode transitions are generated, they are checked to ensure a minimum separation between any two values. If the minimum separation between any two speeds is below $3000/M$ RPM, then all speeds are discarded and the set is generated again;
- The computation time C^m of each mode m is defined as $C^m = U^m \Theta / w^m$. If the generated computation times are not monotonically increasing with respect to modes, then they are discarded, and a new set is generated.

$U = U^P + U^*$ is the overall utilization of the set of periodic and AVR tasks. Task priorities are assigned according to the Rate Monotonic order (i.e., the lower the period the higher the priority), where the period of the AVR task is considered as $T^* = \Theta / \omega^{max}$, that is, its lowest possible inter-arrival time.

In the following experiments we denote as

- EXACT - The analysis presented in this paper (in Section 4);

- ILP - The analysis proposed in [8] using the revised ILP constraints of [9].

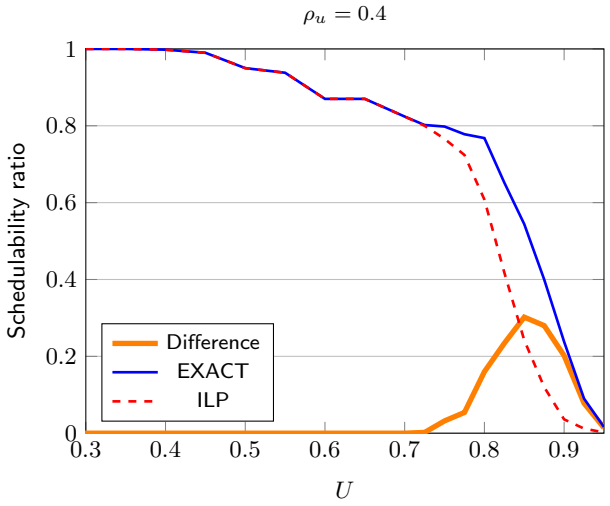

Figure 10: Schedulability ratio as a function of the system load U for $\rho_u = 0.4$.

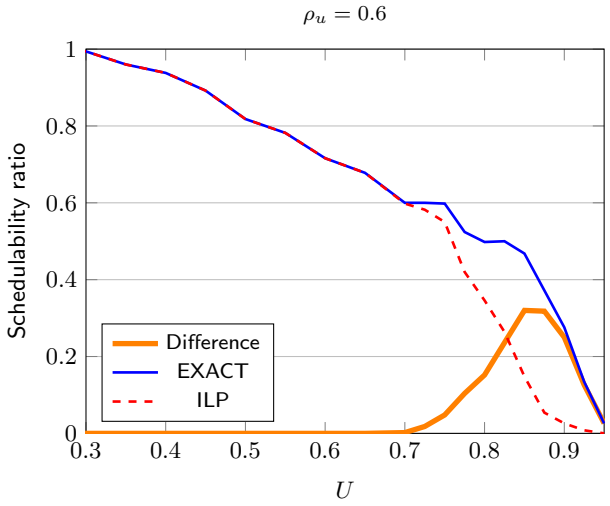

Figure 11: Schedulability ratio as a function of the system load U for $\rho_u = 0.6$.

5.2 Experiment 1

The first experiment was carried out to measure the schedulability ratio of the two analyses as a function of the overall utilization U for task sets composed of $n = 5$ periodic tasks and an AVR task with $M^{min} = 4$ and $M^{max} = 8$. The utilization of the AVR task was computed as $U^* = \rho_u U$. For each value of the utilization, the two schedulability tests were executed over 500 randomly generated task sets.

Figure 10 shows the results of this experiment when the utilization U varies from 0.3 to 0.95, and for $\rho_u = 0.4$. Clearly, both the tests tend to degrade as the system load increases. In the range $[0.7, 0.95]$ the EXACT analysis improves the schedulability with respect to the ILP test, being able to admit 6 times more task sets for $U = 0.9$. Figure 11 shows the results of a similar experiment carried out for $\rho_u = 0.6$, where the gain in schedulability of the EXACT test over ILP is 10 times more for $U = 0.9$. Both figures also report the difference of the two schedulability ratios to better appreciate the results.

Note that the achieved improvement of the proposed analysis exactly occurs in the workload range where these applications typical operate (80% utilization or higher).

5.3 Experiment 2

A second experiment was carried out to better evaluate the dependency of the schedulability ratio on the utilization of the AVR task by varying the factor $\rho_u = U^*/U$ from 0.05 to 0.95. For each value of ρ_u, the two schedulability tests (EXACT and ILP) were executed over 500 randomly generated task sets composed of $n = 5$ periodic tasks and an AVR task with $M^{min} = 4$ and $M^{max} = 8$.

The results of this experiment for $U = 0.85$ are reported in Figure 12. As visible from the plots, the gain in schedulability of the EXACT test increases with ρ_u, until reaching a significant improvement around $\rho_u = 0.6$. Notice that, for high values of the AVR utilization ($\rho_u > 0.6$) the ILP analysis shows a saturation effect in the schedulability ratio, whereas the EXACT test is still able to admit 2-3 times more task sets than ILP.

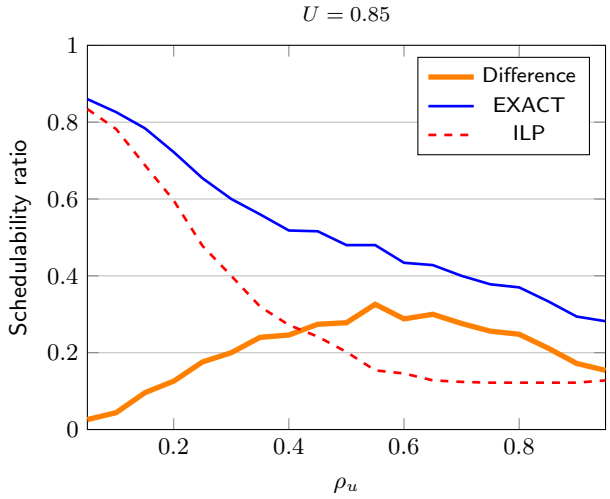

Figure 12: Schedulability ratio as a function of $\rho_u = U^*/U$.

5.4 Experiment 3

In this experiment we measured the schedulability ratio of the two methods by varying the number of modes $M = M^{min} = M^{max}$ of the AVR task. The overall utilization of the task set is fixed to $U = 0.85$, and the utilization of the AVR task is $U^* = 0.4U$. For each value of M, 500 task sets have been randomly generated including $n = 5$ periodic tasks.

Figure 13 shows the results of this experiment, when M is varied from 3 to 12. Note that both the tests decrease their performance as M increases, but the improvement of the EXACT test over ILP increases with the number of modes M, or equivalently, with the number of AVR tasks, as observed at the beginning of Section 5.1.

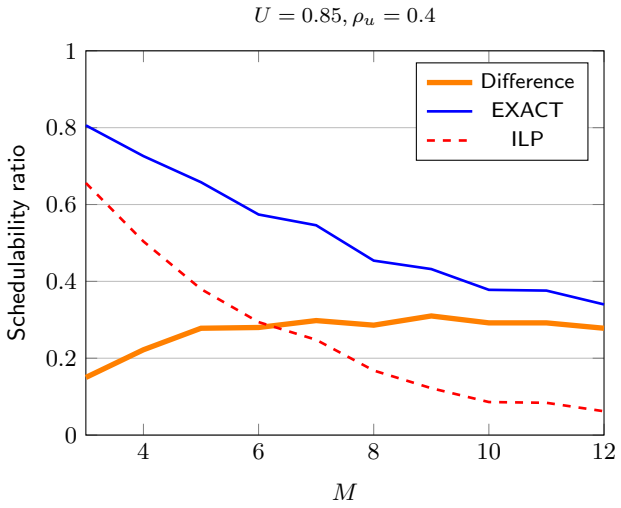

Figure 13: Schedulability ratio as a function of the number of modes M for the AVR task, with $U = 0.85$ and $\rho_u = 0.4$.

6. CONCLUSIONS AND FUTURE WORK

This paper presented the first exact analysis for task sets consisting of periodic tasks and adaptive variable rate tasks with a common activation source, all scheduled by a fixed priority algorithm. This result allows a designer to precisely analyze the behavior of engine control applications in the temporal domain, providing a method for predicting possible overload conditions that could jeopardize the system performance. Simulation results show that the proposed approach always dominates the previous sufficient tests, with significant improvements in terms of schedulability for high processor workloads (80% utilization or higher), which represent the typical operating conditions of engine control applications.

As a future work, we plan to extend the response time analysis to sets with multiple AVR tasks with different angular periods and phases and possibly different independent activation sources. We also aim at using our analysis methodology in a design synthesis process to support the application developer in selecting the mode transition speeds given the implementation of the control functions for each mode.

Acknowledgements

The authors like to thank Martin Stigge for the interesting exchange of ideas on AVR and digraph real-time task models, and Robert I. Davis, Timo Feld, Victor Pollex, and Frank Slomka for the fruitful discussions on schedulability analysis of AVR tasks.

7. REFERENCES

[1] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[2] 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.

[3] G. Buttazzo, L. Abeni, and G. Lipari. Elastic task model for adaptive rate control. In *IEEE Real Time System Symposium*, Madrid, Spain, December 1998.

[4] 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.

[5] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, March 2002.

[6] G. C. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26, January 2005.

[7] 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.

[8] 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 15-17, 2014.

[9] R. I. Davis, T. Feld, V. Pollex, and F. Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *University of York, Department of Computer Science Technical Report, YCS-2014-488*, January 2014.

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

[11] 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 17-19, 2012.

[12] 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.

[13] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, March 2004.

[14] L. Sha, R. Rajkumar, J. P. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264, December 1989.

[15] M. Stigge and W. Yi. Combinatorial abstraction refinement for feasibility analysis. In *Proceedings of the 34nd IEEE Real-Time Systems Symposium (RTSS 2013)*, 2013.

[16] M. Stigge and W. Yi. Refinement-based exact response-time analysis. In *Proc. 26th Euromicro Conference on Real-Time Systems (ECRTS'14)*, 2014.

[17] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or EDF scheduling. In *Proceedings of the Design, Automation and Test Conference in Europe (DATE 2009)*, Nice, France, April 20-24, 2009.