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

Alessandro Biondi, Marco Di Natale, Giorgio Buttazzo
Scuola Superiore Sant'Anna
Pisa, Italy
{alessandro.biondi, marco.dinatale, 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 as a set of modes that switch at given rotation speeds to adapt the overall computational demand. For this reason, these tasks are referred to as adaptive variable-rate (AVR) tasks. This paper presents an exact response time analysis for engine control applications consisting of periodic and AVR tasks scheduled by a fixed priority algorithm. The proposed analysis is first presented for task sets having 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.

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 itself 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 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 unbounded delays on other control activities, causing the loss 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 and cannot be captured by a finite set of modes. 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 having 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 schedulability analysis under fixed priorities, 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 for guaranteeing safety.

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:

- $\theta$ the current rotation angle of the crankshaft;
- $\omega$ the current angular speed of the crankshaft;
- $\alpha$ the current angular acceleration of the crankshaft.

The rotation speed $\omega$ is assumed to be limited within a range $[\omega_{\text{min}}, \omega_{\text{max}}]$ and the acceleration $\alpha$ 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 $\Gamma_P$, whereas the subset of angular tasks is denoted by $\Gamma_A$, 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 $\tau_i^\ast$.

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 $\omega$.

An angular task $\tau_i^\ast$ is characterized by an **angular period** $\Theta_i$ and an **angular phase** $\Phi_i$, 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 $\omega$ and can be expressed as

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

An angular task $\tau_i^\ast$ is also characterized by a relative **angular deadline** $\Delta_i$, expressed as a fraction $\delta_i$ 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 $\Phi_i$ 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 $\tau_i^\ast$ is typically implemented as a set $M_i$ 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 $\tau_i^\ast$ is characterized by a WCET $C_i^m$ and is valid in a speed range $[\omega_{\text{min}}^m, \omega_{\text{max}}^m]$, where $\omega_{\text{min}}^m = \omega_{\text{min}}^i$ and $\omega_{\text{max}}^m = \omega_{\text{max}}^i$. Hence, the set of modes of task $\tau_i^\ast$ can be expressed as

$$M_i = \{C_i^m, \omega_i^m\}, \quad m = 1, 2, \ldots, M_i.$$  

The computation time of a generic AVR job $J_{i,k}$ can be
expressed as a non-increasing step function $C_i$ of the instantaneous speed $\omega$ at its release, that is,

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

An example of $C$ function is illustrated in Figure 1.

$$C_i(\omega)$$

![Graph of C function](image)

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 $(\omega_k, \alpha_k)$ 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 $\alpha$) as:

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

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

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

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

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

If a job $J_{i,k}$ is released when the engine has an instantaneous speed $\omega_k$, the interarrival time $\tilde{T}_1(\omega_k, \omega_{k+1})$ to the next job $J_{i,k+1}$ released with instantaneous speed $\omega_{k+1}$ (if reachable with the acceleration bounds), can be obtained by Equation (3), substituting $\alpha_k$ from Equation (5), which gives:

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

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

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

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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_i$</td>
<td>$i^{th}$ periodic task</td>
</tr>
<tr>
<td>$\tau^*_i$</td>
<td>$i^{th}$ AVR task</td>
</tr>
<tr>
<td>$C_i(\omega)$</td>
<td>WCET of $\tau^*_i$ as a function of the instantaneous speed</td>
</tr>
<tr>
<td>$C^m_i$</td>
<td>WCET of mode $m$ of $\tau^*_i$</td>
</tr>
<tr>
<td>$\omega^m_{i,k}$</td>
<td>Maximum speed for mode $m$ of $\tau^*_i$</td>
</tr>
<tr>
<td>$\Theta_i$</td>
<td>Angular period of $\tau^*_i$</td>
</tr>
<tr>
<td>$T_i(\omega_k, \alpha_k)$</td>
<td>Inter-arrival time between the $k^{th}$ and $(k+1)^{th}$ job instances, assuming the acceleration is $\alpha_k$</td>
</tr>
<tr>
<td>$D_i(\omega_k)$</td>
<td>Relative (temporal) deadline for a job released at speed $\omega_k$</td>
</tr>
<tr>
<td>$\Omega_i(\omega_k, \alpha_k)$</td>
<td>Engine speed at the release of job $J_{i,k}$</td>
</tr>
<tr>
<td>$\tilde{T}<em>i(\omega_k, \omega</em>{k+1})$</td>
<td>Inter-arrival time between a job released at speed $\omega_k$ and the following at speed $\omega_{k+1}$</td>
</tr>
<tr>
<td>$\tilde{T}_i^m(\omega_k)$</td>
<td>Minimum inter-arrival time between a job released at speed $\omega_k$ and the following in mode $m$</td>
</tr>
<tr>
<td>$S(t)$</td>
<td>Set of job sequences for an AVR task</td>
</tr>
<tr>
<td>$C_S(t)$</td>
<td>Set of critical job sequences for an AVR task</td>
</tr>
<tr>
<td>$I(t)$</td>
<td>Interference of a job sequence $s$ of an AVR task</td>
</tr>
<tr>
<td>$I_I(t)$</td>
<td>Interference envelope of an AVR task</td>
</tr>
<tr>
<td>$R_s(\tau_i)$</td>
<td>Response time of task $\tau_i$ interfered by the job sequence $s$ of an AVR task</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Upper-bound for the response time of task $\tau_i$</td>
</tr>
</tbody>
</table>

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 $\tau^*$ released at time $t = 0$ with speed $\omega_0$, as illustrated in Figure 2, and suppose that the job executes for its WCET $C(\omega_0)$. 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 $\alpha^+$ and would occur after $T(\omega_0, \alpha^+)$ time units, whereas the latest job release corresponds to the maximum deceleration $\alpha^-$, 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 $\omega_1(t)$ 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 $\omega_1$ 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.
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. Being the speed domain 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.

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

Each sequence \( \omega, \alpha \in \text{S}(\tau) \) generates an interference \( I(\omega, \alpha, t) \) representing 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 \( \omega \) and \( \Pi \) is the sum of all the computational request imposed by the previous jobs. The procedure 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 \( \omega_0 \) 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 \( \tau^* \) is composed of a set of consecutive released jobs \( J_0, \ldots, J_n \), and a set of speeds \( \omega_0, \ldots, \omega_n \), where each element \( \omega_k, k = 0, \ldots, n \) 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 \( \text{S}(t) \) be such a set. Intuitively, each path in the search tree represents a job sequence.

Each sequence \( s \in \text{S}(t) \) generates an interference \( I(s, t) \), which is a function of \( \omega_0 \) and \( \omega_k, k = 1, \ldots, n \), because it depends on the speed evolution pattern experienced by the AVR task. In general, \( I(s, t) \) can be expressed as

\[
I(s, t) = C(\omega_0) + \sum_{k=1}^{n} C(\omega_k) \text{step} \left( t - \sum_{j=1}^{k} T(\omega_{j-1}, \omega_j) \right),
\]

where

\[
\text{step}(x) = \begin{cases} 
1 & \text{if } x \geq 0 \\
0 & \text{if } x < 0 
\end{cases}
\]

In addition, it is convenient to define the partial contribution \( I(s, i, \omega) \) to \( I(s, t) \) starting from the intermediate nodes (corresponding to job activations) of the job tree. \( I(s, i, \omega) \) 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 \( \omega \).
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 speeds 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 \( J_t \) of an AVR task \( \tau^* \), the set of its dominant speeds \( W = \{ \omega_{t,0}^d, \omega_{t,1}^d, \ldots \} \) is the set of engine speeds at the release time of \( J_t \), such that for any other engine speed \( \omega \notin W \) there exists at least one \( \omega_{t,j}^d \in W \) such that \( I^{(s)}(t, i, \omega_{t,j}^d) \geq I^{(s)}(t, i, \omega) \).

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 \( \tau^* \) 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

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

Based on this result, the worst-case interference caused by an AVR task \( \tau^* \) 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 \( D_t \leq T_* \). 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 \( \tau_i \) interfered by a set of regular periodic tasks and a single AVR task \( \tau^* \), all having higher priority than \( \tau_i \). In the following, \( hp(t) \) denotes the set of periodic tasks having higher priority than \( \tau_i \). To simplify the notation, the set \( \mathcal{CS} \) is used as a shortcut for \( \mathcal{CS}(D_t) \), denoting the set of critical job sequences in the interval \([0, D_t]\).

Given a particular job sequence \( s \) of \( \tau^* \), the response time of \( \tau_i \), denoted as \( R_i^{(s)} \), can be expressed as

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

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 \( \tau_i \) can be computed only considering the critical job sequences in the interval \([0, D_t]\).

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

\[
R_i = \max_{s \in \mathcal{CS}} R_i^{(s)}.
\]  

**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 \( \tau^* \), defined as

\[
I(t) = \max_{s \in \mathcal{CS}} I^{(s)}(t).
\]

and computing a response time upper bound \( \overline{R_i} \) as

\[
\overline{R_i} = \min \left\{ t \mid C_i + I(t) + \sum_{j \in hp(t)} \left\lceil \frac{t}{T_j} \right\rceil C_j = t \right\}.
\]

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 \( \tau_e \) executing with an AVR task \( \tau^* \) having higher priority. Also, assume that
the set $\mathcal{CS}(D_1)$ is composed by only two sequences, i.e., $\mathcal{CS}(D_1) = \{s_a, s_b\}$. 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 $\tau^*$ 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.](image)

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 $R_t$ is originated from different sequences $s \in \mathcal{CS}$ depending on the interference of the higher priority tasks. Hence, contrarily to classical periodic tasks, in order to obtain the characterization of the exact response-time it is not possible to abstract the interference of $\tau^*$ 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 $\tau_i$ 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 $R_t$, it is necessary to identify all possible critical job sequences $s \in \mathcal{CS}$ and then compute the response-time $R^*_t$ for each sequence $s$.

In this section, we present an algorithm to efficiently compute $R_t$ 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, $\text{ResponseTime}$, computes the contribution to the interference of one additional job activation. $\text{ResponseTime}$ 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 $\omega$ (at the time the job is activated), the current time $t$, and the execution time requests $\Pi$ (the contribution to the interference) accumulated up to time $t$.

Each time $\text{ResponseTime}$ is called, $\omega$ is one of the dominant speeds and $t$ the candidate point in time for the job activation. Given $t$ and $\omega$, 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 $\omega_1$ in the figure) are considered by the algorithm, while those occurring after the idle time (such as $\omega_2$ and $\omega_3$ 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 $\text{singleJobRT}$). 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 $\text{getDominants}(\omega)$ 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 $\omega_1$), 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 $\omega_{id}$ in the figure), they are considered for the next recursive call, otherwise (such as $\omega_p$ in the figure), they are pruned.

![Figure 6: Pruning and branching in the algorithm for computing the response time.](image)

Figure 8 shows the algorithm to check the schedulability of $\tau_2$ using the procedure $\text{ResponseTime}$. Procedure $\text{SchedulabilityTest}$ starts by computing the initial dominant speeds in the full range allowed for engine speeds, (i.e., $\left[\omega^{\min}, \omega^{\max}\right]$). Then, for each initial dominant speed $\omega_0$ we compute the response-time candidates: since all the candidates represent response-time values related to possible job sequences starting from $\omega_0$, the maximum $R$ of such candidates is taken as response time for speed $\omega_0$. If the response-time $R$ exceeds the deadline $D_t$, then $\tau_2$ is not schedulable; otherwise, if $R$ results lower (or equal) than $D_t$ for each initial dominant speeds, then $\tau_2$ is schedulable.
1: procedure ResponseTime(i, ω, Π, t)
2:    idleTime ← GetIdleTime(i, ω, Π, t);
3:    jobRT ← SingleJobRT(i, ω, idleTime);
4:    RTCandidates.add(jobRT);
5:    for convenience, the interference from the tasks in the AVR
6:    having the same angular period and phase (assumed as 0
7:    exactly at the same time instant, the definition of
8:    when considering engine-control application).
9:    for convenience (as shown in
10:    surprising, the additional pruning in the search allowed
11:    idle-time detection makes the computation of the exact
12:    interference of the tasks in the
13:    the intereference of the tasks in the
14:    not change. The intereference of the tasks in the
15:    the set of jobs from the AVR tasks are activated ex-
16:    I
17:    by idle-time detection makes the computation of the exact
18:    critical in-
19:    a finite set of modes for all the AVR tasks). Each mode m of
20:    defined by the corresponding speed range (ω^m, ω^m)
21:    and a worst case execution time
22:    C^m(ω) = \sum_{j \in hp^\alpha(\tau^\star)} C_j(\omega^m).
23:    At this point, the approach presented in the previous section
24:    can be applied.
25: 4.2 Response-Time of an AVR Task Interferred
26:    by Periodic Tasks
27:    Let us now consider the response-time of an AVR task τ^\star
28:    interfered by periodic tasks (assuming there are periodic tasks
29:    and computing the response time for each
30:    considering each mode of τ^\star, so obtaining a response-time
31:    yes we have to verify
32:    with m = 1, . . . , M.
33:    Finally, to check the schedulability of τ^\star we have to verify that
34:    R^m \leq D(\omega^m) for each mode m = 1, . . . , M.
35: 4.3 Response Time of an AVR Task Interfered
36:    by other AVR Tasks
37:    We now address the schedulability of an AVR task τ^\star
38:    interfered by both periodic tasks and multiple AVR tasks having
39:    the same angular period are considered, only one job for
40:    each high-priority AVR task can produce interference, i.e., a
41:    single computation time must be accounted for. We denote
42:    as the set of AVR tasks having higher priority than τ^\star.
43:    Now, the dependency on the speed ω_0 can be removed by
44:    \{ t | C(ω_0) + \sum_{i \in hp^\alpha(\tau^\star)} \left\lceil \frac{t}{T_i} \right\rceil C_i = t \},
45:    where hp(τ^\star) denotes the set of periodic tasks having higher
46:    \{ t | C^m + \sum_{i \in hp^\alpha(\tau^\star)} \left\lceil \frac{t}{T_i} \right\rceil C_i = t \},
47:    with m = 1, . . . , M.
48:
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 using synthetic workloads. 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 quantization 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 $\alpha^+ = -\alpha^- = 1.62 \cdot 10^{-4}$ rev/msec².

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 task is generated as follows:

- The utilization $U_i$ of each task $\tau_i$ 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 $[3, 100] \text{msec}$;
- 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 $U_{max} = U^*$. The utilization $U_{max}$ 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 $\omega_{max}$ of each mode $m < M$ is randomly generated in the range $[1000, 6000] \text{RPM}$. The maximum speed for mode 1 is always set at the maximum speed $\omega_{max}$. 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 $3000/M \text{RPM}$, then all speeds are discarded and the set is generated again;
- The computation time $C_i$ of each mode $m$ is defined as $C_i = U_i \omega_{i}/\omega_{max}$. 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 $\rho_u = 0.4$.](image)

![Figure 11: Schedulability ratio as a function of the system load $U$ for $\rho_u = 0.6$.](image)

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 $M^\text{min} = 4$ and $M^\text{max} = 8$. 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 $\rho_u$, 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 $M^\text{min} = 4$ and $M^\text{max} = 8$.

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 $\rho_u$, 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 $\rho_u = U^*/U$.](image)

5.4 Experiment 3

In this experiment we measured the schedulability ratio of the two methods by varying the number of modes $M = M^\text{min} = M^\text{max}$ 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.
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