Proc. of the 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems, Trento, Italy, September 5, 2017.

A framework for the co-simulation of engine controls and task scheduling

Paolo Pazzaglia, Marco Di Natale, Giorgio Buttazzo, Matteo Secchiari

Scuola Superiore Sant'Anna, Pisa, Italy
{paolo.pazzaglia,marco.dinatale,giorgio.buttazzo}@sssup.it
msecchiari@gmail.com

Abstract. To evaluate the impact of scheduling latency and task design on the performance of engine control applications, we developed a co-simulation framework, based on Simulink and an extension of the T-Res scheduling simulator tool. The objective of the research and the tool development is to provide a better characterization of the very popular problem of scheduling and analysis of Adaptive Variable Rate Tasks (AVR) in engine control. The purpose of the tool is to go beyond the simplistic model that assumes hard deadlines for all tasks and to study the impact of scheduling decisions (and possibly missed deadlines) with respect to the functional implementations of the control algorithms and the engine performance. The developments include a co-simulation framework and a set of models for the engine components in order to evaluate the performance with respect to fuel efficiency, consumption, soot and NOx emissions.

1 Introduction

The study of the schedulability conditions for engine control tasks (denominated as adaptive variable rate - AVR [1]) has become popular in the real-time research community because of the novel nature of the problem, which applies to the concept of Cyber-physical systems; the special activation conditions that apply to some of the system tasks and the adaptive nature of the computations.

Several engine control tasks are not periodic or sporadic, but are activated by the rotation of the engine crankshaft (a parameter of the physical controlled system). In addition, to compensate for the increased CPU load at high rotation speeds (and more frequent activation times), the code implementation of these tasks is defined in such a way that at given speed boundaries, the implementation is simplified and the execution time is reduced. A typical engine control application consists of time-driven periodic tasks with fixed periods, typically between a few milliseconds and 100 ms (see [2], page 152), and angular tasks triggered at specific crankshaft angles.

The activation rate of such angular tasks varies with the engine speed (variable-rate tasks). For example, for speeds from 500 to 6500 revolutions per minute (RPM), the interarrival times of the angular tasks range from about 10 to 120 ms (assuming a single activation per cycle).

With respect to the set of activation instants, the dependency from a physical phenomenon characterizes this problem as truly belonging to the class of problems in cyber-physical systems (CPS). However, in many papers the dependency of the timing and scheduling problem from the physics of the controlled system is restricted to the set of activation events and every other concern is hidden under the typical assumption of hard deadlines.

In reality, this problem (as many others) is representative of a class of control systems in which deadlines can be missed without catastrophic consequences, and the problem should actually be defined as a design optimization, where the objective is to select the controls implementations and the scheduling policy in such a way that a set of engine performance functions are optimized (including power, emissions, noise, pollution). These performance functions depend in complex ways from timing parameters, such as jitter and latency. Informally, the objective of the scheduler is not to miss too many deadlines or produce actuation signals that are too much delayed.

Formally, the problem is quite complex and extremely unlikely to be solved in a simple, closed analytical form or even with a general procedure for expressing the dependency of the performance from scheduling. This is the reason for the investigation of alternative approaches that are based on the simulation of the three system components in a joint environment:

- A model of the engine combustion process (the physical system or plant)
- A model of the engine controls
- A model of the task configuration and execution and of the scheduler.

In this work we describe the framework that was developed for this task; the components developed for modeling the AVR tasks and the engine subsystems of interest; and the early results obtained from simulations. This framework significantly improves our previous work [3], especially in the combustion modeling and control parts. The co-simulation framework is based on the popular Simulink tool from the Mathworks and a simulator of real-time scheduling and task execution for single-and multicore platforms: the T-Res framework.

In the following sections, the problem is introduced (in Section 2). Then, in Section 3 we outline the main components and functionality of the framework, and in Section 4 we briefly introduce the T-Res framework. The extensions to the TRes custom components are outlined in Section 5. The Engine model and controller components are described in Section 6. The preliminary analysis results are in Section 7, and the related work is in Section 9.

2 The problem

One of the main objectives of a fuel injection system is to determine the point(s) in time and the quantity of fuel to be injected in the cylinders of an engine, relative to the position of each piston, which is in turn a function of the angular position of the crankshaft. In a reciprocating engine, a common reference is the *top dead centre* (TDC), that is, the position in which one of the pistons is nearest to the crankshaft (Figure 1). In a four-cylinder engine, the pistons are paired

in phase opposition, so that, when two of them are in a TDC, the others are in the bottom dead center or BDC. The TDC is the typical reference point for the functions and actions that need to take place within the rotation. These action include (among others) computing the phase (time relative to the TDC) of the injection and the quantity of fuel to be injected, but also checking whether the combustion occurred properly.

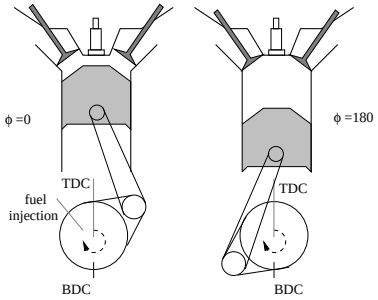

Fig. 1. Relationship among engine phases and reference points in the crankshaft rotation period. In a 4-cylinder engine, cylinder pairs are in phase opposition.

The time between two activations (at the TDC) is not constant, nor arbitrary, but depends on the rotation speed of the engine, which can vary within given ranges with a given maximum acceleration. At low revolution rates, the time interval between two reference points (the TDC for a set of cylinders) is large and allows the execution of sophisticated controls and possibly multiple fuel injections. The same algorithm cannot be executed at higher revolution rates, because it would lead to an overload, generating several deadline misses. Therefore, the implementation is adapted using a simplified algorithm that reduces the (worst-case) execution time (i.e., the functions to be executed) when the rotation speed falls within pre-defined ranges. For most cars, the rotation speed typically varies between 600 and 6000 revolutions per minute (rpm), which maps to activation intervals between 100 and 10 ms.

The model proposed to describe such a type of engine control tasks is referred to as *Adaptive Variable Rate* task model, or AVR-model [1].

The analysis of the schedulability conditions for AVR tasks has been conducted by assuming that tasks have hard deadlines, often implicit, meaning that each task must complete before its next activation. This is not necessarily true. Besides evidence gathered from the common industrial practice, an indication that this is indeed the case is that the AUTOSAR automotive standard (and its predecessor OSEK) allow system configurations in which multiple instances of the same task are active at the same time.

If deadline misses are allowed, it is important to understand what are the consequences of a late task and what action (if any) should be performed by the scheduler. In our model, we assume that control tasks program a TPU (Time Processing Unit) that is in charge of actuating the fuel injectors. In this case,

the effect of a missed deadline is that the TPU remains programmed with the parameters (angle of injection and duration of the injection) computed in the previous cycle.

An additional objective of our framework is to explore scheduling options, including the use of the Earliest Deadline First scheduling policy in the context of applications of this type. Finally, we aim at verifying the conjecture that the engine performance functions can be fit with an exponential function. This assumption was used to derive an algorithm for computing the optimal mode switching speeds in [4]. Overall, the parameters of interest that are meant to be captured by the model are **(i)** the engine thermodynamic efficiency, **(ii)** the NOx emissions and **(iii)** the soot emissions.

3 The Simulation Framework for the Performance Analysis of Controls and Scheduling Design

To explore the impact of scheduling on the engine performance we constructed a co-simulation framework in which the master simulation engine is Simulink. The engine model has been defined as a Simulink model, which will be presented in detail in Section 6. Next, we leveraged the T-Res cosimulation environment for the co-simulation of the task execution and scheduling [5]. An introduction to T-Res is presented in Section 4, while some extensions of the framework are described in Section 5.

For the development of the engine model we leveraged information from several sources, including engine models for the steady state and event-based models as described in [6] and other empirical models found online.

The engine controls have been incrementally defined and now include several components that allow to simulate individual cylinders, with injectors, the turbocharger, the intake and exhaust manifolds, and the gas recirculation. The desired angle of injection and the injection time are defined by calibration tables obtained from the literature.

4 The T-Res cosimulation framework

The T-Res (Time and Resource-aware simulator) framework is the result of a project that aims at integrating in Simulink the effect of code execution latencies and scheduling delays. T-Res consists of a set of custom Simulink blocks representing tasks and kernels and allows to interface the Simulink simulation engine, acting as master, with a scheduling simulator in a co-simulation environment (see Figure 2). The scheduling simulator (we use RTSim [7], but the backend simulation engine can be changed) computes the scheduling delays and holds the output valuess of the corresponding tasks until their simulated completion time. This allows to simulate delays in the production of output values and the corresponding impact on the control function.

T-Res provides a custom block for representing the kernel and its scheduler (left side of Figure 3). The block is configured with the selection of the scheduling

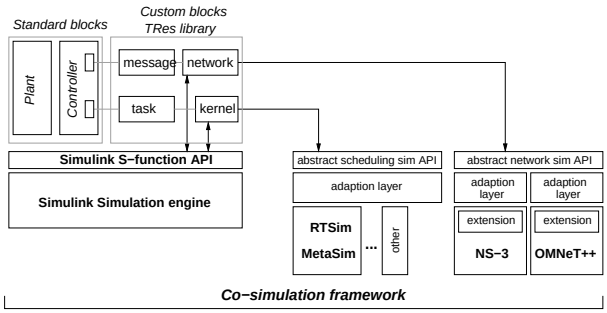

Fig. 2. The TRes cosimulation architecture.

policy and the behavior in case of deadline (period) overrun. The possible options are to drop the task or to let it execute until its late completion. The kernel block provides a set of activation signals as output for the tasks it manages.

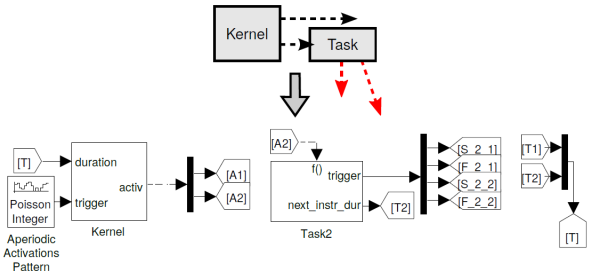

Fig. 3. The custom block for schedulers in T-Res.

These activation signals go to instances of the second type of custom blocks, representing tasks (right side of Figure 3). Each task receives an activation signal from the kernel (indicating when the task begins or resumes execution), and is characterized by an execution time estimate (a configuration parameter), and a signal going back to the kenel and providing the amount of time that is still required by the task at each point in time. The execution time of a task is defined in T-Res as a sequence of instructions of predefined execution (constant or computed from a distribution) using a simple language and assigned as one of the task block parameters. The task block produces as output a set of activation and latch signals for all the functional subsystems that are executed by the task. With respect to the activation, sporadic tasks are characterized by an activation event going as input to the kernel block, or a periodic activation specification, provided as a configuration parameter to the kernel (for details, refer to [5]).

5 Extending T-Res for modeling AVR tasks

For the purpose of this project we extended the task model block and the timing information associated with it to allow the representation of Adaptive Variable Rate tasks.

For what concerns the activations, the AVR task can be treated as a particular case of aperiodic task. The task block in T-Res includes a signal for the explicit activation in case of event-triggered tasks. This signal is used to define the activation of the task in correspondence to given angular positions of the engine crankshaft, as defined by a simple trigger block which is activated at specific angles. For what concerns the execution, an *ad-hoc* AVR task block must be created for handling the mode switching: this AVR task block has a dedicated *mode* input that is referred to the active mode index, as presented in Figure 4. The mode input can be used for multiple purposes, and in particular it is used to select which control functionality must be executed. For the engine control system, each execution mode represents the need of having different execution times, associated with different control strategies: the most complex and complete control tasks (which are time-consuming) will be activated at low speeds, while the simpler (and time-saving) ones will be used at higher speeds.

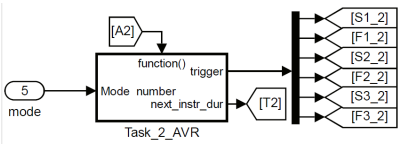

Fig. 4. A custom block for modeling an AVR task.

The AVR block triggers an arbitrary number of pairs of segment-latch blocks, each implementing a set of actions, as the classic task block. The segment block contains the control instructions and it is triggered at the start of *execution* of the job. On the other hand, the latch block holds the previous data until it is triggered at the *completion* of the job, thus releasing the new data from the segment to the model.

For each modality it is possible to specify which pairs of blocks are executed or skipped, and also specify different execution times. The implementation of the AVR segment-latch blocks that has been used for this paper is presented in Figure 5. This particular implementation has been created for modelling a multiple injection control: the first segment block computes the main injection parameters, the second the post-injection and the third the pre-injection parameters. The strength of this approach is that the activation of the second and third pair can be selectively disabled when the injection mode is changed, without losing information because intermediate outputs are sensed after each latch block.

The mode index is provided by an external block called *mode selector* which senses the current engine speed and selects the execution modality that is required, according to a user-defined map. Each mode is activated when the engine

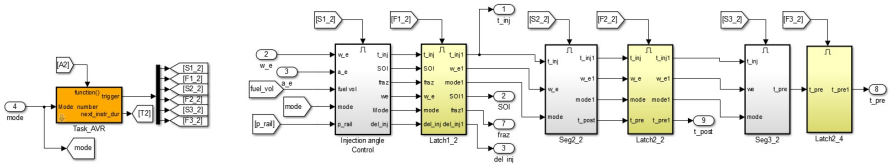

Fig. 5. An example of AVR task modelled with TRES, implementing the multiple injection control

speed value lies in a specific interval of switching speeds. The optimal selection of the switching speeds is a non-trivial process, and a performance-based analysis on the topic has been provided in [4]. The mode selector supports also hysteresis between the switching, in order to avoid potentially dangerous oscillations when the engine operates near the switching speeds.

5.1 Variable deadline implementation

In order to correctly test the system under EDF scheduling, the deadline of the AVR task must be changed at every job activation, accordingly to its activation rate. The deadline value, which in the original TRes implementation is statically defined as a workspace value, must then be treated as a variable.

A formulation for the relative deadline of an AVR task has been calculated in [8] and is reported here:

$$D_i(\omega) = \delta_i\big(\sqrt{\omega^2 + 2\Theta_i\alpha} - \omega\big)/\alpha, \tag{1}$$

where ω is the engine speed, δ_i a fraction of angular period ($\delta \in [0,1]$), Θ_i the angular period and α the engine acceleration, supposed constant during the angular interval Θ_i. However, if this formula is implemented as it is in a real ECU, it implies iteratively solving on-line calculations, which could excessively increase the overhead of the scheduler. The possible solutions for this problem are: **(i)** creating a sufficiently detailed look-up table for different (ω, α) combinations, or **(ii)** using an approximated computation. In order to reduce the number of inputs in the Kernel block (thus reducing the overall complexity of the scheduler) we decided to use an approximate computation, where the deadline at every iteration k has the value:

$$\begin{cases} D_i(0) = \delta_i T_0 \\ D_i(k) = \delta_i\big(t_{act}(k) - t_{act}(k-1)\big) \end{cases} \tag{2}$$

where T_0 is the (precomputed) interval between two activations at the starting speed, and $t_{act}(k)$ is the absolute time of the k-th job activation. The deadline is then directly provided to the external scheduler RTSIM at the job arrival.

6 Simulink Models of the Engine and the Control Tasks

Figure 6 shows the model of the engine and the control functionality in Simulink. The blocks in the upper part of the figure represent the engine subsystems that

are currently considered and include a turbocharger, compressor manifold, intercooler, intake and exhaust manifolds and the model of the engine cylinders. The subsystem on the bottom part of the figure wraps our model of the engine controller, with its outputs: the injection angle and duration and the VGT.

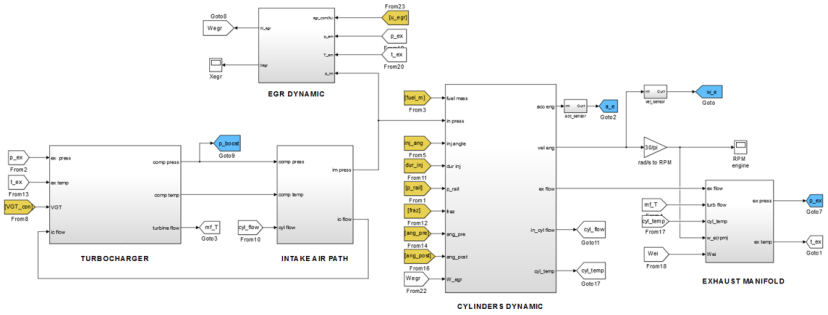

Fig. 6. Physical part of the engine model implemented in Simulink

The framework has been improved since the one presented in our previous work [3], both in the physical and control parts. The engine now supports the simulation of multiple cylinders, properly phased, instead of a mean-modelling with a single cylinder. This choice allows to precisely capture the influence of control errors which are split on different cylinders.

6.1 The engine model

The main improvements of the physical model are related to the cylinder dynamics and the analysis of the effect of multiple injections in the same cycle. A generic cylinder block has been created, containing all the mechanical and combustion dynamics, as a Simulink library block, with the possibility to consider a large number of status variables of the internal dynamics. An arbitrary number of cylinders can then be easily added to the model, by only changing the initial conditions and angular phasing. The equations used for this framework have been carefully implemented using the state-of-the-art models, mainly from the work of Kiencke and Nielsen [9]. In detail, the block is composed by:

- valves dynamics, modelled as variable area flow restrictions, in order to have a more realistic model of the intake and exhaust flow through the cylinders;
- injector dynamics, controlling directly the timing of the nozzle opening and the rail pressure, in order to manage multiple-injection strategies;
- energy release of combustion using the zero-dimensional Chmela model proposed in [10], which assures a good sensitivity to the shape of the fuel injection profile, while maintaining low computational requirements;
- piston dynamics as a crankshaft-rod mechanism;

- pressure and temperature dynamics, with equations of energy balance, wall losses and flow exchanges
- NOx formation dynamics, modelled using the semi-empirical equation from the work of Guardiola et al. [11];
- soot formation, with both formation (Hiroyasu model [12]) and oxidation (Nagle and Strickland-Constable model [13]) dynamics;
- thermodynamic efficiency of the entire combustion cycle, computed as the ratio of generated work to the heat released by the combustion;

The model presents also the dynamics of the crankshaft and engine friction, a Variable Geometry Turbocharger, the air path of intake and exhaust gas, and the dynamics of the temperature for these blocks. Finally, the model has also been enriched with an Exhaust Gas Recirculation block for reducing the NOx emission. A general overview of the physical model is presented in Figure 6.

6.2 The engine control

A control block has been developed to take care of the most important dynamics of the engine. The control uses data from sensors (engine speed, acceleration, intake air flow, boost pressure, exhaust pressure and pedal input) to compute the actuation signals. The control variables are the turbine palettes orientation and a mix of injection-related variables: rail pressure, injection angles, injection duration and fraction of fuel. The control laws are implemented using both lookup tables and classic controllers such as PID blocks.

For what concerns the injection control, three strategies have been implemented: triple injection (pre-main-post), double injection (either pre-main or main-post) and single injection. The reason behind this choice is that splitting the injection with different shapes provides better results in terms of combustion of fuel and reduction of pollutants. In particular, pre-injections are used to have a better initial mixture and a more uniform combustion, while post-injection are typically used to burn the residual fuel.

The framework is able to capture changing of performance in NOx, soot and thermodynamic efficiency by varying a great number of parameters, such as starting angle of injection, duration of injections, fraction of fuel, relative angle between splits and so on.

The model consists of a kernel, and seven tasks. One of the seven tasks is an AVR, three are periodic controllers and two represents other computations.

As the modelled engine has 4-cylinders and the injection must be performed once every two cycles for each cylinder, the AVR task is activated every half rotation. The TPU is actuated after 180^o of the AVR task activation, and that instant corresponds also to the AVR deadline.

7 Objective and Status

A detailed modeling of the control function is necessary to better understand the impact of deadline misses or long latencies. Depending on the implementation of

the control function, a deadline miss may result in a late actuation, or a missed actuation or even an actuation with old data. In our controls implementation, the AVR task computes the phase and duration of the injection and passes them to the task that simulates the injection actuators. Hence, a missed deadline results in actuating the injectors with the values computed in the previous cycle with a likely error in phase and duration with respect to the ideal values.

The objective of our framework is multifold:

- To understand the effect of the scheduling on the engine performance and to use the environment for analyzing the impact of scheduling policies and parameters, such as evaluating fixed priority vs EDF or different possible priority assignments and task configurations.
- To analyze the timing parameters that truly of interest for evaluating the performance of the engine and possibly attempt a characterization that isolates the attributes of interest. This includes, among others, the evaluation of schemes like m-k deadline misses, or overload management (maximum lateness).
- To better characterize the design problem consisting in the optimal selection of the transition speeds for AVR tasks.

The final application goal of this work is to create a tool for testing different strategies of control and scheduling before implementing them in real cars, which can be particularly useful especially for automotive companies. However, currently the scope of the analysis is relatively limited, due to the huge complexity of both the physical model and the real software implementation. The difference among the engine control laws at different speeds (one for each possible execution mode of the AVR tasks) mostly consists in the possibility of defining one, two or three fuel injections. We assume at low speeds three injections are possible, while at high rotation speeds there is only enough computation time available to compute the angle and duration of a single injection.

The current engine model is capable of simulating several performance functions of interest for a variable number of injections. Figure 7 shows the simulated amount of NOx pollutants produced for simulation runs in which one (green line), two (red) or three (blu line) injections are performed during the cycle. The figure shows the benefit on the pollutant emissions that can be obtained by using multiple injections in the cycle.

Similarly, the graph in Figure 8 shows the quantity of soot produced in the simulated runs under the hypothesis of one, two or three injections for each cycle.

8 Studying the effects of deadline misses on performance

In Diesel engine control, the injector actuation is managed by the Time Processing Unit (TPU), or by its most recent version called Enhanced Time Processing Unit (eTPU). The TPU is a co-microprocessor which works in synchronous modality, driven by both an internal clock and an Angle Clock that is based on the crankshaft angular position. Among all its functions, it generates the voltage signal for the injectors, using the control data provided by the CPU. This

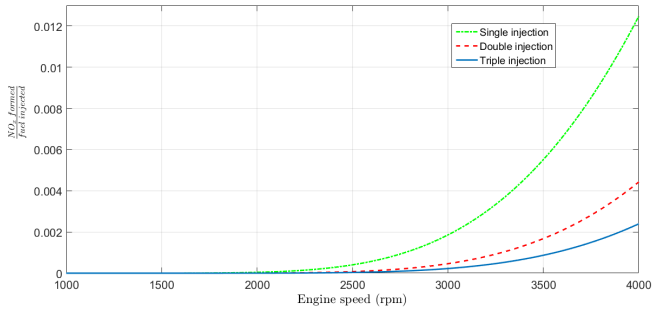

Fig. 7. Decrease in NOx emission with multiple injections

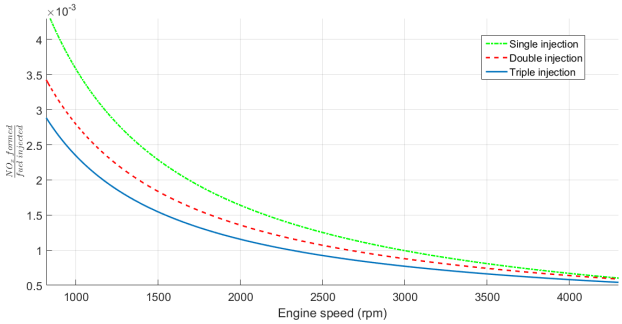

Fig. 8. Decrease in soot production with multiple injections

action is done with high precision at very specific angles, synchronized with the crankshaft rotation.

It is essential that the control of injection provides the computed values before the TPU injection process is activated. For this reason, this instant has been defined as the deadline of the AVR task: if the AVR task misses its deadline and cannot provide new data to the TPU, the TPU must and does use the old data for the next injection. However, this effect can produce errors in the timing of injection (i.e., the starting angle) and in the quantity of fuel injected, if the old data are not good for the next cycle, for example if the engine suddenly accelerates. In this case, the errors can be significant and worsen the combustion performance. Thus, in order to minimize emissions of pollutants and wear on the engine components, having a model to understand how scheduling delays are related to performance is of critical importance.

Currently, within the assumptions of our model, the simulation is able to show how the scheduling delays result in errors in the angle/duration of the injection actuation and the corresponding loss in performance (for each selected metric). Our objective is to relate the errors in phase and duration of the injection to a possible loss of efficiency or an increase in pollutants, providing ways to analyze the impact of scheduling with respect to the first performance function

of interest. Figure 9 shows the results of an example run in which the task execution times have been defined to have one or two deadline misses on each cycle (for one or two cylinders) under a throttle input consisting of a ramp (to force a variation in the engine rpm and the ideal injection conditions).

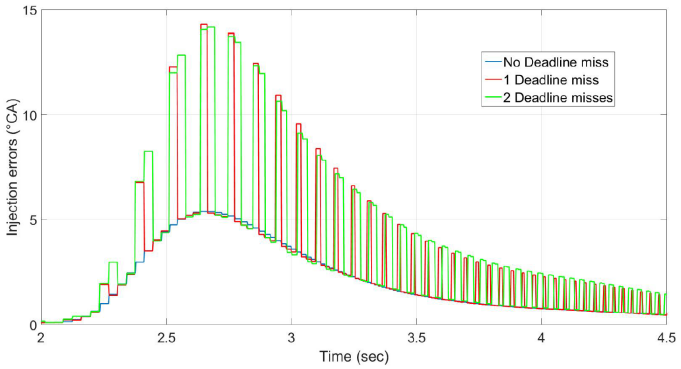

Fig. 9. Angular error because of deadline misses

In the figure graph, the vertical axis shows the phase error in the actuation of the injection for a sample manoeuvre consisting of a sudden acceleration and a corresponding increase in the engine rotation speed from low to high values. Three graphs are plot in the figure. The graph in blue (dark) color shows the angle error when the execution time of the AVR task does not generate any deadline miss. When forcing a single miss (red line) or two misses on each cycle (green line) the injection angle error grows to almost 15 degrees.

The angular error in the injection is related to a variation (loss) in the power performance of the engine. Figure 10 shows the corresponding effect of the deadline misses imposed on the system (in the same experiment configuration) on the thermodynamic efficiency. The graph clearly shows how deadline misses result in a loss of efficiency. Similar results are obtained for the performance that relate to the pollutants NOx and soot.

9 Related Work

The presentation of the task model in which engine control tasks are implemented with a variable computational requirements for increasing speeds is in [14],

These tasks are also referred to adaptive variable-rate (AVR). Analyzing the schedulability of tasks sets consisting of both periodic and AVR tasks is a difficult problem that has been addressed by several authors under various simplifying assumptions, under both fixed priority scheduling [15–17] and Earliest Deadline First (EDF) [18–20]. Other authors proposed methods for computing the exact interference [1] and the exact response time [17] of AVR tasks under fixed priority scheduling. It has been shown [20] that, given the large range of possible

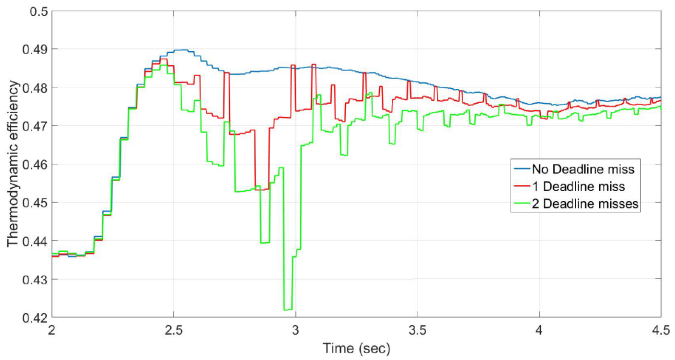

Fig. 10. Performance loss because of deadline misses

interarrival times of an AVR task, fixed priority scheduling is not the best choice for engine control systems since, while EDF exhibits a nearly optimal scheduling performance. Based on this fact, Apuzzo et al. [21] provided an operating system support for AVR tasks under the Erika Enterprise kernel [22].

All the papers considered above, however, focus on the schedulability analysis of task sets consisting of periodic and AVR tasks, without any concern on engine performance. A performance-driven design approach has been addressed in [23] for finding the transition speeds that trigger the mode changes of an AVR task.

A very large number of projects target the evaluation of scheduling policies and the analysis of task implementations. A necessarily incomplete list includes Yartiss [24], ARTISST [25], Cheddar [26], and Stress [27].

Finally, TrueTime [28] is a *freeware*[1] Matlab/Simulink-based simulation tool that has been developed at Lund University since 1999. It provides models of multi-tasking real-time kernels and networks that can be used in simulation models for networked embedded control systems. TrueTime is used by many research groups worldwide to study the (simulated) impact of lateness and deadline misses on controls. In TrueTime, the model of task code is represented by *code functions* that are written in either Matlab or C++ code. Several research works investigate the consequences of computation (scheduling) and communication delays on controls. An overview on the subject can be found in [29].

10 Conclusions and Future Work

The co-simulation framework is in its early developing stages and starts producing results of interest. However, to better characterize the performance impact of scheduling delays, a better model of the control tasks and of some engine characteristics is still needed. In the context of the project we also aim at the evaluation of different task allocation strategies on multicore platforms.

[1] `http://www3.control.lth.se/truetime/LICENSE.txt`

References

1. 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.
2. L. Guzzella and C. Onder, *Introduction to modeling and control of internal combustion engine systems*. Springer Science & Business Media, 2009.
3. P. Pazzaglia, A. Biondi, M. Di Natale, and G. Buttazzo, "A simulation framework to analyze the scheduling of avr tasks with respect to engine performance."
4. A. Biondi, M. Di Natale, and G. Buttazzo, "Performance-driven design of engine control tasks," in *Proceedings of the 7th International Conference on Cyber-Physical Systems*. IEEE Press, 2016, p. 45.
5. F. Cremona, M. Morelli, and M. Di Natale, "Tres: a modular representation of schedulers, tasks, and messages to control simulations in simulink," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, Salamanca, Spain, April 13, 17 2015, pp. 1940–1947.
6. L. Guzzella and C. H. Onder, *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer-Verlag, 2010.
7. L. Palopoli, G. Lipari, L. Abeni, M. D. Natale, P. Ancilotti, and F. Conticelli, "A tool for simulation and fast prototyping of embedded control systems," in *LCTES/OM*, S. Hong and S. Pande, Eds. ACM, 2001, pp. 73–81.
8. A. Biondi and G. Buttazzo, "Engine control: Task modeling and analysis," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 525–530.
9. U. Kiencke and L. Nielsen, "Automotive control systems: for engine, driveline, and vehicle," 2000.
10. F. G. Chmela and G. C. Orthaber, "Rate of heat release prediction for direct injection diesel engines based on purely mixing controlled combustion," SAE technical paper, Tech. Rep., 1999.
11. C. Guardiola, J. López, J. Martin, and D. Garcia-Sarmiento, "Semiempirical in-cylinder pressure based model for no x prediction oriented to control applications," *Applied Thermal Engineering*, vol. 31, no. 16, pp. 3275–3286, 2011.
12. H. Hiroyasu and T. Kadota, "Models for combustion and formation of nitric oxide and soot in direct injection diesel engines," SAE Technical Paper, Tech. Rep., 1976.
13. J. Nagle and R. Strickland-Constable, "Oxidation of carbon between 1000-2000 c," in *Proceedings of the fifth carbon conference*, vol. 1, no. 1. Pergamon Press London, 1962, p. 154.
14. 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.
15. 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)*, Beijing, China, April 2012, pp. 28–38.
16. R. I. Davis, T. Feld, V. Pollex, and F. Slomka, "Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling," in *Proc. 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, Berlin, Germany, April 2014.
17. A. Biondi, M. D. Natale, and G. Buttazzo, "Response-time analysis for real-time tasks in engine control applications," in *Proceedings of the 6th International Conference on Cyber-Physical Systems (ICCPS 2015)*, Seattle, Washington, USA, April 14-16, 2015.

18. 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.
19. A. Biondi and G. Buttazzo, "Engine control: Task modeling and analysis," in *Proc. of the International Conference on Design, Automation and Test in Europe (DATE 2015)*, Grenoble, France, March 9-13, 2015, pp. 525–530.
20. A. Biondi, G. Buttazzo, and S. Simoncelli, "Feasibility analysis of engine control tasks under EDF scheduling," in *Proc. of the 27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 8-10, 2015.
21. V. A. A. Biondi and G. Buttazzo, "OSEK-like kernel support for engine control applications under EDF scheduling," in *Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2016)*, Vienna, Austria, April 11-14, 2016.
22. "ERIKA enterprise: an OSEK compliant real-time kernel," http://erika.tuxfamily.org/drupal/.
23. A. Biondi, M. D. Natale, and G. Buttazzo, "Performance-driven design of engine control tasks," in *Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS 2016)*, Vienna, Austria, April 11-14, 2016.
24. Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, M. Qamhieh *et al.*, "Yartiss: A tool to visualize, test, compare and evaluate real-time scheduling algorithms," in *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2012, pp. 21–26.
25. D. Decotigny and I. Puaut, "Artisst: an extensible and modular simulation tool for real-time systems," in *Object-Oriented Real-Time Distributed Computing, 2002.(ISORC 2002). Proceedings. Fifth IEEE International Symposium on.* IEEE, 2002, pp. 365–372.
26. F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *ACM SIGAda Ada Letters*, vol. 24, no. 4. ACM, 2004, pp. 1–8.
27. N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Stress: A simulator for hard real-time systems," *Software: Practice and Experience*, vol. 24, no. 6, pp. 543–564, 1994.
28. A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, "How does control timing affect performance?" *IEEE control systems magazine*, vol. 23, no. 3, pp. 16–30, 2003.
29. K. J. Astrom and B. Wittenmark, "Adaptive control," in *Prentice Hall*, 2016.