Towards co-simulation of embedded platforms and physics-based models

Towards Co-Simulation of Embedded Platforms and Physics-Based Models

Yon Vanommeslaeghe*‡, Paul De Meulenaere*‡, Joachim Denil*‡,
Francesco Cosco†§, Bart Forrier†§, Jan Croes†§
yon.vanommeslaeghe@uantwerpen.be, paul.demeulenaere@uantwerpen.be, joachim.denil@uantwerpen.be,
francesco.cosco@kuleuven.be, bart.forrier@kuleuven.be, jan.croes@kuleuven.be

*CoSys-Lab (FTI), University of Antwerp, Belgium
†Department of Mechanical Engineering, KU Leuven, Belgium
‡AnSyMo/CoSys, Flanders Make, Belgium
§DMMS, Flanders Make, Belgium

***Abstract*—The embedded software in some classes of contemporary cyber-physical systems (CPS) is required to run models of its physical environment in real time and with high accuracy and precision. The question to which extent the embedded software can achieve these requirements largely depends on the properties of the embedded platform (i.e. embedded hardware and middleware). In this paper, we focus on the modelling of some of the properties of an embedded platform and on the co-simulation of these properties with the physics-based models. This will result in an assessment of the influence of the embedded properties on the performance of the physics-based models in early stages of the development of embedded software for CPS.**

I. Introduction

Along with the growth in intelligent behaviour of current CPS, their embedded software shows increasing 'awareness' of their physical environment, and more in particular of the physical plant that they are supposed to control. Therefore, models of the physical environment often need to be part of the embedded software. Well-known examples are virtual sensing, model-predictive control and self-calibration.

Executing these physics-based models on embedded hardware is not always straightforward. In a typical design process, the physics-based models are designed by mechanical or control engineers with the purpose to be run in a simulation environment, mainly aiming at achieving a good correlation with their physical counterparts. When re-using such models on an embedded processor, problems can be expected related to real-time performance, memory consumption, numerical precision, I/O-properties, etc. Bertsch et al. [1] anticipate some of these problems when transitioning from a simulation environment to an embedded environment in the context of the Functional Mockup-Unit (FMU).

Karsai et al. [2] argue that the design of CPS can only be accomplished in a co-design fashion and that modeling the embedded execution platform needs to take part in it.

This brings us to the point that it is beneficial to incorporate the proper embedded platform properties into the simulation models of the CPS under design. One of the well-known tools in this respect is TrueTime [3], which allows for the incorporation of execution delays into Simulink models. Denil et al. [4] show that the DEVS-modelling language is also suitable to express the embedded delays and to simulate their performance impact.

This work encourages us to further investigate the co-simulation of models of processor-based embedded platforms with models of their applications, especially for those CPS-applications for which the performance is presumably influenced by the embedded system architecture. While TrueTime can be used for this purpose, it requires parts of the model to be implemented in MATLAB-scripts. In a co-design setting, the model to be deployed is often provided, e.g. in Simulink. In this case re-implementation of the model in MATLAB would be counterproductive. Additionally, when the new model including the embedded platform properties is to be used by the original engineer to optimize their design, it is favourable to make minimal modifications to the original implementation. For these reasons, we rely on the combination of Simulink and SimEvents. While Simulink is appropriate to describe the continuous models of the physics that take part in the application, SimEvents in turn is an interesting candidate to model the discrete event properties of the embedded platform architecture [5]. This includes the modelling and simulation of the scheduling of application tasks on one or more processors [6].

The current paper focusses on a virtual sensing case based on an Extended Kalman Filter (EKF) which comprises a model of the physical plant under observation. This use case is further described in the next section. Having an embedded realisation of this Kalman filter in mind, Section III explores which performance characteristics are deteriorated by which embedded components. It shows simulation results on the performance drawbacks of certain choices of the embedded platform configuration. Finally, in Section V, we discuss some future ideas to setup a more generic co-simulation framework for this type of problems.

II. Motivating Example

As an example case we make use of a virtual torque sensor developed by Forrier et al. [7]. This virtual sensor is a model-

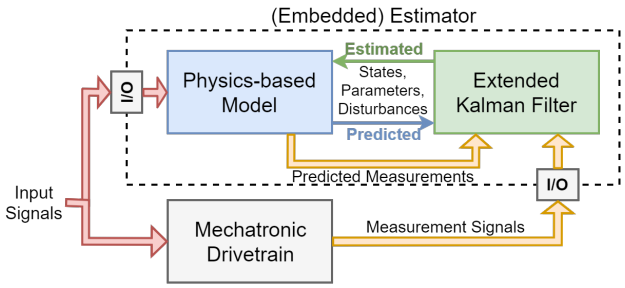

Fig. 1: General diagram of a model-based estimator.

based estimator. Figure 1 shows a general overview of the workings of such an estimator. Certain inputs, in this case phase voltages, are applied to the actual drivetrain, after which values such as currents, accelerations, etc., can be measured. These same inputs are also applied to a model of the drivetrain. This model is then used to predict the measured values, as well as internal states, parameters, etc. The difference between the predicted and actual measurements is then used by the Kalman filter to correct the internal states, among other things. This is an recurrent process whereby the model is expected to converge to the actual driveline over time. This allows variables that are difficult to measure, such as the external load torque, to be extracted from the model. The test setup of the drivetrain is presented in detail in [8]. Note that, in our experiments, we use this estimator with simulated measurements, i.e. generated from a model, as data recorded on the actual test-setup was unavailable at the time.

III. Method

Bariş et al. [9] identify three major factors that impact the precision of a physics-based estimator after deployment. However, they mostly focus on the first factor: precision loss due to the use of single- instead of double-precision floating point numbers. Two other factors, additive measurement noise and update frequency (timing), are not explored. It is these factors that are investigated in the following sections.

A. Additive Measurement Noise

Additive measurement noise is, as the name implies, noise that is added to the measurements that serve as inputs for the estimator. There are multiple factors that contribute to the noise: electromagnetic interference (EMI) and possible crosstalk on the analog inputs, quantization noise due to the resolution of the analog to digital converter (ADC), and accuracy of the ADC. But also other factors such as a possible gain or offset error, or possible non-linearity of the ADC.

To investigate the impact these factors have on the accuracy of the estimator, they are modelled in Simulink as separate subsystems, as illustrated in Figure 2. Varying the parameters of this interface model, such as ADC-resolution, offset, gain, etc., allows both the embedded and the control engineers to determine the impact of the previously mentioned factors.

To test the impact of the modelled interface, two copies of the same estimator are placed in a Simulink model. The interface model is placed on the inputs of one of the estimators to mimic an estimator running on an embedded system. The other copy of the estimator serves as a reference. Measurements (voltages, currents, etc.) are then fed to the two estimators. The corresponding reference torque is subtracted from the estimated torques to obtain their respective error. This setup is illustrated in Figure 3.

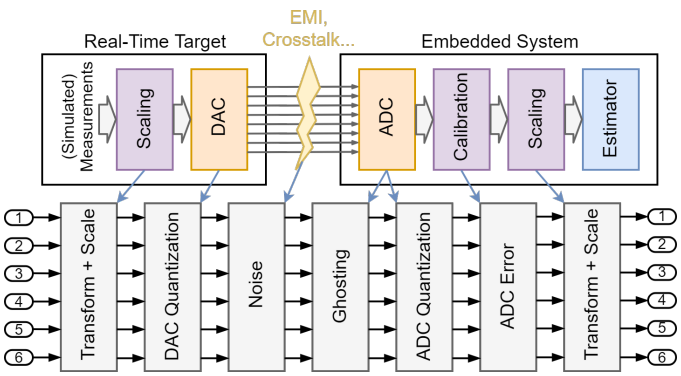

Fig. 2: Different parts of the interface are modelled as different subsystems.

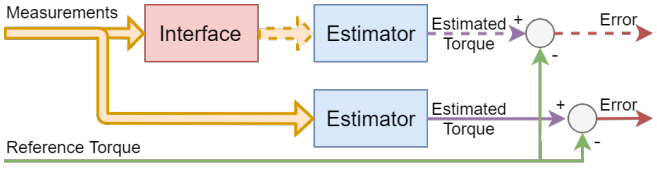

Fig. 3: Diagram showing test setup for interface model.

A MATLAB script is used to vary different parameters of the input model, run the simulations and process the results. The mean and standard deviation over the different runs can then be plotted to further investigate the impact on the output.

B. Timing

In general, the achievable accuracy of an estimator depends on its update frequency. For simple, single step estimators determining the maximum frequency is rather straightforward as it is almost entirely dependent on its worst case execution time (WCET). However, when CPS become more complex, the software running them may need to be split into multiple tasks. In this case, task interaction becomes important as it can have an impact on timing and subsequently on the behaviour of the system. This was demonstrated by Morelli et al. [10], who saw a difference in the behaviour of a controller depending on the priorities of its different tasks.

To be able to simulate the impact of these timings on the behaviour of the system, these effects need to be added to the original model. As previously mentioned, SimEvents allows for the modeling of the scheduler of an embedded system. SimEvents models can be mixed with Simulink models containing the application, thus forming a hybrid model. The link between the scheduler and the control loop can then be made

using function triggered subsytems and function calls. This setup is illustrated in Figure 4.

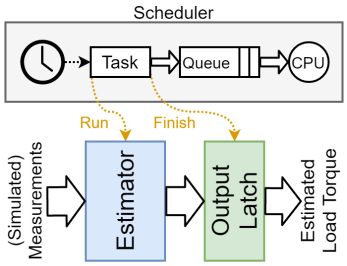

Fig. 4: Illustration of test setup for timing.

To test the usefulness of SimEvents in simulating timing behaviour, a small test case was made starting from an example implementation of a scheduler [6]. The example was modified to have a single task with a period of 250 µs. This corresponds to the specified update rate of our estimator case.

The estimator is converted into a triggered subsystem. Simulink simulations follow the Zero Execution Time (ZET) model. This means that when the subsystem is triggered, its outputs are calculated based on its current inputs and are updated in the same timestep. The execution time of the model on the processor has no impact on the simulated time. This is in contrast with the actual behaviour after deployment, where the outputs would only become available after a certain amount of time (the execution time). To simulate this behaviour, another triggered subsystem is added to the outputs of the estimator to serve as an output latch [10]. When triggered, this subsystem copies its inputs to its outputs.

When the task defined in SimEvents is activated, the estimator subsystem is triggered and its outputs are calculated. When the specified execution time has passed, the task is finished and the output latch is triggered. This makes the latest outputs available to the rest of the model. The output of the estimator can then be compared to the reference torque in the same way as described in the previous subsection.

A MATLAB script is used to set the execution time, run the simulation and collect the outputs. The execution time was varied between 190 µs and 260 µs. Additionally, as the real life execution time is not constant, a uniform random value between 0 µs and 50 µs is added to the set execution time for each time the task is executed. These values were chosen based on the actual distribution of execution times we observed during previous experiments. As the fixed part of the execution time is increased, this random part means that a certain percentage of deadlines will be missed, delaying the execution of the next task.

IV. Results

A. Additive Measurement Noise

Figure 5 shows a plot with the accuracy of the estimators as a function of the ADC-resolution. The blue line is the mean error for the simulated embedded estimator including

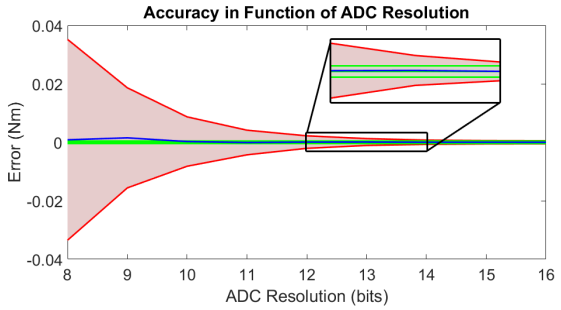

Fig. 5: Estimator accuracy in function of ADC resolution (no noise).

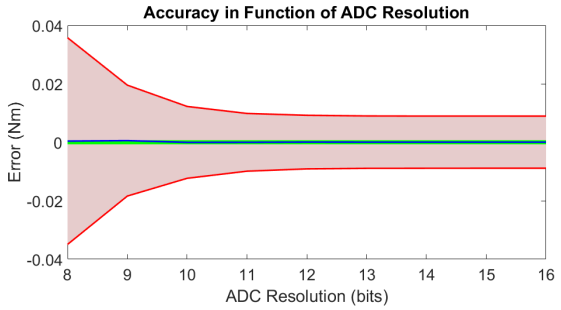

Fig. 6: Estimator accuracy in function of ADC resolution (with AWGN).

the ADC-model; the red band is the corresponding confidence band, i.e. the same mean ± the corresponding standard deviation. The green confidence band refers to the ideal estimator. The inset shows this more clearly. This graph shows that as the ADC-resolution increases, the red band converges to the green band, as we would expect. The same experiment is also performed with additive white gaussian noise (AWGN) added to the measurements. The results of this are shown in Figure 6. Here, a resolution of 12-bit is close to the standard deviation of the AWGN. The graph shows that increasing the resolution above this point no longer has a noticeable impact on the accuracy of the output.

B. Timing

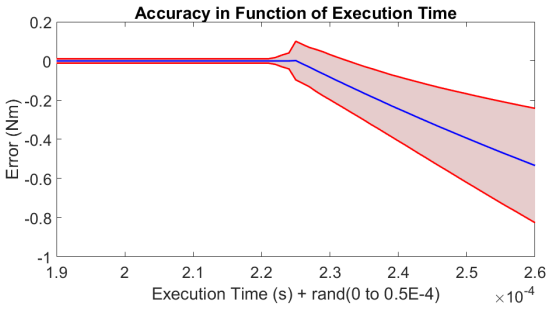

Fig. 7: Estimator accuracy in function of execution time.

Figure 7 shows the accuracy in function of the execution time. This graph shows that the system can tolerate a certain

amount of deadline misses. When the fixed part of the execution time is set to 220 µs, about 40 % of deadlines are missed, yet there is no noticeable decrease in (numerical) accuracy. It is only when the execution time is increased further that the accuracy of the estimator starts to decrease.

Normally deadline misses would be avoided at all costs, but this scenario does show the possibilities of using SimEvents to introduce scheduling behaviour in Simulink models.

V. Conclusions and Future Work

By explicitly modelling the interface of the embedded system and adding this to the original model, it becomes possible to simulate the impact of a certain hardware aspects of the embedded system. This allows both the control engineer and the embedded engineer to investigate the effects of certain parameters. This has several advantages. It allows the control engineer to possibly relax or restrict certain requirements. For example, if increasing the ADC resolution above a certain point yields a negligible increase in performance, a less precise ADC could be used in the final deployment. This in turn decreases the overall cost and complexity of the system. Alternatively, the control engineer could add an additional requirement that shielding should be used on all signal wires, to decrease the noise due to EMI. Additionally, it allows the control engineer to modify their design to deal with the increased noise (e.g. update noise covariance matrices, add filtering, etc.) if this is deemed necessary.

The use of SimEvents to introduce timing behaviour in Simulink models looks promising, especially in a co-design context, as it makes the effect of timing on the control behaviour of the model visible to the control engineer. This becomes more important when working with more complex systems and complex embedded platforms. For example, if a model has to be divided into multiple tasks, task interaction becomes important as it will have an effect on the timing behaviour. For this reason the presented methodology should be expanded to include more properties of a Real-Time Operating System, such as preemption, task interaction, etc.

AADL (Architecture Analysis & Design Language) [11] already offers a modelling language to describe embedded platforms. In our future work, we intend to use AADL to express the platform configuration and to couple these models with SimEvents such that the AADL-models become executable. This idea is expressed in Figure 8. The triplet AADL, SimEvent and Simulink would hence offer a generic modelling and simulation environment to assess the performance of computationally intensive algorithms on embedded platforms. This idea could even be extended towards other modelling languages. In the latter case, the use of the Functional Mockup Interface (FMI) standard is appealing to couple the different models. However, in such case a large part of the discrete time properties of the embedded platform (such as scheduling operations) will most probably need to be implemented in the FMI-master. An automatic generation of the FMI-master [12] would serve these needs.

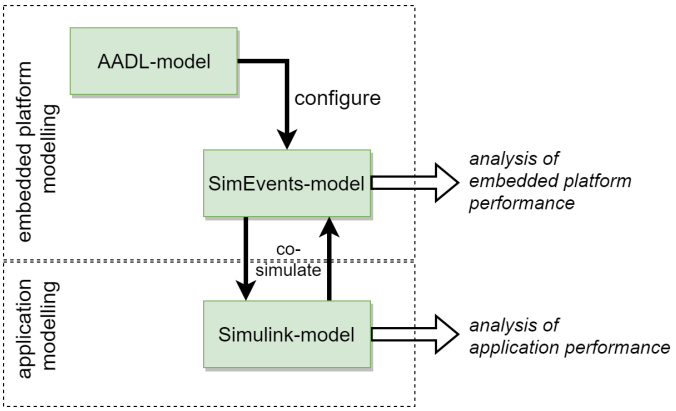

Fig. 8: AADL as configuration for SimEvents.

VI. Acknowledgements

This research was supported by Flanders Make, the strategic research centre for the manufacturing industry in Belgium, within the Model based Force Measurements (MoForM) project.

References

[1] C. Bertsch, J. Neudorfer, E. Ahle, S. S. Arumugham, K. Ramachandran, and A. Thuy, "Fmi for physical models on automotive embedded targets," in *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*, no. 118. Linköping University Electronic Press, 2015, pp. 43–50.

[2] G. Karsai and J. Sztipanovits, "Model-integrated development of cyber-physical systems," in *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*. Springer, 2008, pp. 46–54.

[3] D. Henriksson, A. Cervin, and K.-E. Årzén, "Truetime: Real-time control system simulation with matlab/simulink," in *Proceedings of the Nordic MATLAB Conference*. Copenhagen, Denmark, 2003.

[4] J. Denil, P. De Meulenaere, S. Demeyer, and H. Vangheluwe, "Devs for autosar-based system deployment modeling and simulation," *Simulation*, vol. 93, no. 6, pp. 489–513, 2017.

[5] M. I. Clune, P. J. Mosterman, and C. G. Cassandras, "Discrete event and hybrid system simulation with simevents," in *Proceedings of the 8th international workshop on discrete event systems*, 2006, pp. 386–387.

[6] W. Li, R. Mani, P. J. Mosterman, and T. Hubscher-Younger, "Simulating a multicore scheduler of real-time control systems in simulink," in *Proceedings of the Summer Computer Simulation Conference*. Society for Computer Simulation International, 2016, p. 11.

[7] B. Forrier, F. Naets, and W. Desmet, "Broadband load torque estimation in mechatronic powertrains using nonlinear kalman filtering," *IEEE Transactions on Industrial Electronics*, 2017.

[8] B. Forrier, R. Boonen, and W. Desmet, "Development of a novel test setup for validation of virtual sensing on mechatronic drivetrains," 2016.

[9] O. Bariş, P. De Meulenaere, J. Steckel, B. Forrier, J. Croes, and W. Desmet, "Model-based physical system deployment on embedded targets with contract-based design," in *Software Engineering and Advanced Applications (SEAA), 2017 43rd Euromicro Conference on*. IEEE, 2017, pp. 296–300.

[10] M. Morelli and M. Di Natale, "Control and scheduling co-design for a simulated quadcopter robot: A model-driven approach," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2014, pp. 49–61.

[11] AS5506, "Architecture analysis & design language (aadl)," *Embedded Computing Systems Committee, SAE*, 2004.

[12] B. Van Acker, J. Denil, H. Vangheluwe, and P. De Meulenaere, "Generation of an optimised master algorithm for fmi co-simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International, 2015, pp. 205–212.