












# Co-simulation of a Model Predictive Control System for Automotive Applications

Cinzia Bernardeschi<sup>1</sup> , Pierpaolo Dini<sup>1</sup> , Andrea Domenici<sup>1</sup>  ,  
Ayoub Mouhagir<sup>2</sup> , Maurizio Palmieri<sup>1</sup> , Sergio Saponara<sup>1</sup> ,  
Tanguy Sassolas<sup>2</sup> , and Lilia Zaourar<sup>2</sup> 

<sup>1</sup> Department of Information Engineering, University of Pisa, Pisa, Italy  
[andrea.domenici@unipi.it](mailto:andrea.domenici@unipi.it)

<sup>2</sup> Université Paris-Saclay, CEA, List, 91120 Palaiseau, France

**Abstract.** Designing a Model Predictive Control system requires an accurate analysis of the interplay among three main components: the plant, the control algorithm, and the processor where the algorithm is executed. A main objective of this analysis is determining if the controller running on the chosen hardware meets the time requirements and response time of the plant. The constraints, in turn, should be met with a satisfactory tradeoff between algorithm complexity and processor performance. To carry out these analyses for an autonomous vehicle control, this paper proposes to leverage parallel co-simulation between the plant, the model predictive controller and the processor.

**Keywords:** Model predictive control · Co-simulation · Autonomous vehicles

## 1 Introduction

Control algorithms based on *model predictive control* (MPC) are increasingly being employed in embedded systems with high-performance requirements and stringent constraints, such as automotive applications. MPC relies on the availability of a mathematical model of the controlled plant, used at each sampling period to evaluate a prediction of the plant's future behaviour over a given timespan (the *prediction horizon*) and choose optimal values for the control variables, to be applied at the next sampling period [13].

Designing an MPC system for embedded applications requires an accurate analysis of the interplay among three main components: the plant, the control

---

A. Domenici—This work has been partially supported by the European Processor Initiative (EPI) project, which has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement № 826647, and by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Department of Excellence).

algorithm, and the processor where the algorithm is executed. With the model-based design approach, the analysis exploits the results of the simulations. In particular, a detailed simulation of the processing architecture executing the control software is needed, given the role of processor performance in meeting real-time constraints. This is a typical situation where co-simulation provides substantial support to developers who need to model subsystems from different areas of expertise: algorithms, processor architecture and plant physics.

This work presents an approach to enable the analysis of MPC systems through co-simulation. To this end, an open source library for MPC algorithm, GRAMPC [10], has been extended with the implementation of a standard interface for co-simulation, FMI (Functional Mock-up Interface) [5]. The MPC algorithm contains a *prediction* model of the plant that is distinct from the *actual* model. This allows the analysis of the controller under a variation of the actual model parameters. The approach also encompasses MPC performance analysis, thanks to the use of the VPSim [7] virtual prototyping tool for complex electronic Systems-on-Chip (SoC) from the SESAM framework [26], which supports FMI co-simulation.

The application of the proposed approach is shown in a case study from autonomous vehicle control, where the plant, the model predictive controller, and the processor are simulated in parallel. The plant is an autonomous car that must reach a destination along a road with a given geometry, avoiding obstacles. The vehicle is simulated with a standard kinematic model implemented in C with the GRAMPC framework, adapting an example from the GRAMPC distribution. The processor architecture is an ARMv8 multi-core processor, simulated in the VPSim framework. The metrics used to analyse different co-simulation runs are the difference of the actual trajectory from the reference one and the execution time of the GRAMPC algorithm, that should be less than the co-simulation step.

The paper is organised as follows: Sect. 2 introduces a selection of related works; background on MPC and the GRAMPC library is briefly reported in Sect. 3; Sect. 4 illustrates the proposed approach for multi-model simulation of automotive systems, while Sect. 5 shows the application to a case study from autonomous driving; finally, Sect. 6 contains conclusions and further work.

## 2 Related Work

Among the many works available to readers looking for an extensive background on wheeled vehicles dynamics, we may cite [14]. More specifically, Yurtsever et al. [28] provide a survey on recent work about autonomous driving. Also the literature on model predictive control offers many fundamental texts, e.g. [13].

In model-driven development, co-simulation [11] can be applied in the analysis of complex cyber-physical systems that integrate a high-level control algorithm with pre-existing closed implementations of lower-level plant dynamics.

Lee et al. [17] report on the co-simulation of an MPC-controlled heating, ventilation and air-conditioning plant, using an ad-hoc Python-based infrastructure to connect a building simulator with a Matlab controller. Similar ad-hoc solutions have been proposed in several works, e.g., von Wissel et al. [27], who use

Simulink S-functions to connect a powertrain model developed on the Siemens LMS Amesim simulator with an MPC controller developed in the Honeywell OnRAMP environment. Using S-functions to couple a Simulink model to different simulators is a common technique, used, e.g., in [4], where a Simulink model of a human heart was coupled to an executable formal model of a pacemaker.

The Functional Mockup Interface 2.0 [5] is a *de facto* standard for co-simulation, and INTO-CPS [16] is an integrated tool chain for model-based design based on FMI.

An FMI infrastructure based on the TISC co-simulation platform was presented by Gräber et al. [12]. In their work, FMUs simulate the plant, an optimizer, and a system estimator. The plant can be modeled with different tools, the optimizer is built with the MUSCOD-II software package using a direct multiple shooting method. A vapor compression cycle is discussed as an application example. An FMI-based infrastructure was used by Ceusters et al. [6], who generate an FMU from a Modelica simulator of multi-energy systems, and use it to communicate with a Python-based environment that models two alternative controllers, one based on MPC and one on reinforcement learning. Another FMI-based framework for co-simulation of human-machine interfaces was presented in [21]. Co-simulation has been paired with formal methods to validate and verify control systems of various kinds [2, 3], including robot vehicles [9, 20].

In the automotive field, the interaction between multi-physics modelling/simulation environments and embedded software development environments has been addressed by many works. Recently, the eFMI (FMI for embedded systems) standard has been proposed as a result of the EMPHYSIS (Embedded systems with physical models in the production code software) project [19].

### 3 Model Predictive Control and the GRAMPC Framework

This section introduces a very succinct description of the concept of model predictive control and of the GRAMPC framework [10] for the simulation of MPC systems.

#### 3.1 Model Predictive Control

A model predictive control system iteratively solves an Optimal Control Problem (OCP) of the following form [10], where  $t \in [0, T]$  is the MPC-internal time coordinate and  $T$  is the prediction horizon:

$$\min_u J(u, x_k) = V(x(T)) + \int_0^T l(x, u, \tau) d\tau \quad (1)$$

$$M\dot{x} = f(x, u, t_k + \tau) \quad (2)$$

$$x(0) = x_k \quad (3)$$

$$x(\tau) \in [x_{\min}, x_{\max}] \quad (4)$$

$$x(T) \in \Omega_\beta \quad (5)$$

$$u(\tau) \in [u_{\min}, u_{\max}], \quad (6)$$

where (1) is the cost functional, which depends on the time evolution of the control variables' vector  $u$  and of the sampled state variables' vector  $x_k$ . The first term ( $V(x(T))$ ) of the cost functional represents the *terminal* cost associated with the final state at the end of the prediction horizon, while the second term represents the *integral* cost computed over the whole trajectory over the prediction horizon. The system dynamics are expressed by (2), where the mass matrix  $M$  defines the inertial properties of the system, and  $t_k = t_0 + k\Delta t$ , with  $0 < \Delta t < T$ , is the  $k$ -th sampling instant.

The state of the system at the beginning of the  $k$ -th control interval is given by (3). The remaining relations express constraints on the state and the control inputs. In particular,  $\Omega_\beta$  is the set of states such that the terminal cost is less than or equal to  $\beta$ . This constraint is typically used to ensure stability.

The controller computes the trajectory of control variables that minimizes (1), and its first segment of simulated length  $\Delta t$  is applied as a plant input during the actual (real-time) control period  $[t_k, t_{k+1})$ .

A common form for the cost functional uses quadratic norms of the form

$$V(x) = \|x - x_{\text{des}}\|_P^2 \quad (7)$$

$$l(x, u) = \|x - x_{\text{des}}\|_Q^2 + \|u - u_{\text{des}}\|_R^2, \quad (8)$$

where  $(x_{\text{des}}, u_{\text{des}})$  is the desired set-point and the norms are weighted by the positive (semi-)definite matrices  $P$ ,  $Q$  and  $R$ , defined according to the application.

### 3.2 The GRAMPC Framework

The GRAMPC (Gradient-Based MPC) framework supports simulation of non-linear systems under MPC by providing a highly configurable optimization algorithm. Users must supply a model of the plant to be simulated, by coding a set of C-language functions implementing well-defined, yet flexible interfaces. Users also set options and parameters to customize the optimization and simulation algorithms. In particular, a user may choose one of a set of available solvers for the optimizer.

The optimization algorithm implements an augmented Lagrangian method, based on the gradient-descent paradigm, also exploited in other adaptive [8] and learning [15] control techniques for modern mechatronic systems. Such an algorithm, at each iteration, requires the evaluation of the plant's dynamics (for the prediction), of the cost functional, of their partial derivatives w.r.t. state and control variables, and of the constraint relations. As mentioned above, all these computations are specified by the user with C functions. For example, function `ffct` computes the plant dynamics, `Vfct` and `lfct` compute the terminal and integral cost, respectively, `dldx` computes the gradient of the integral cost w.r.t. the state variables, and `hfct` checks the inequality constraints.

The implementation of the optimization and simulation algorithm maintains a structure (`grampc`) that contains all the information of the problem at hand (including the current state of the plant, the values of the command variables, the time). A function named `grampc_run` takes the `grampc` structure as an input and executes a step of the MPC algorithm, updating `grampc`.

In order to execute a simulation, a user writes a source file with the functions modeling the system (`ffct` etc.) and a file with a main program where parameters and options are initialized. Then, a loop starts, invoking `grampc_run` at each iteration. The resulting values of control and state variables can be further processed and printed out.

Figure 1 summarises how a simulation is built on top of the GRAMPC framework. The framework is composed of a library providing the implementations of the core MPC functions, and the declarations of the interfaces to be implemented by the user, who provides the problem formulation in two source files, one with the prediction plant model and the other with the initialization and the main loop. It may be observed that, in the GRAMPC framework, it is not possible to simulate the controlled plant and the controller separately, since the controlled plant model coincides with the prediction plant model used in the controller.

## 4 Proposed Approach

This work is based on the idea of embedding a GRAMPC model into an FMU written in C where the time advancing function `fmi2DoStep` invokes the `grampc_run` function. The control values updated in `grampc` are forwarded as FMI output variables and the current state stored in `grampc` is overwritten by the FMI input variables as shown in Fig. 2. This is achieved by exploiting a pre-existent FMU generator such as [18] or [22] to create the basic FMU structure that should be compiled together with the whole GRAMPC library source files and the two GRAMPC files of the system at hand.

The file with the model of the plant does not require changes, while the file with the implementation of the algorithm requires some minor changes: the initialization of the GRAMPC parameters should be wrapped into a function that will be invoked by the FMU initialization function `fmi2SetupExperiment`, while the code for executing the MPC algorithm should be wrapped in a function that will be invoked by the `fmi2DoStep` function. Finally, the values stored in

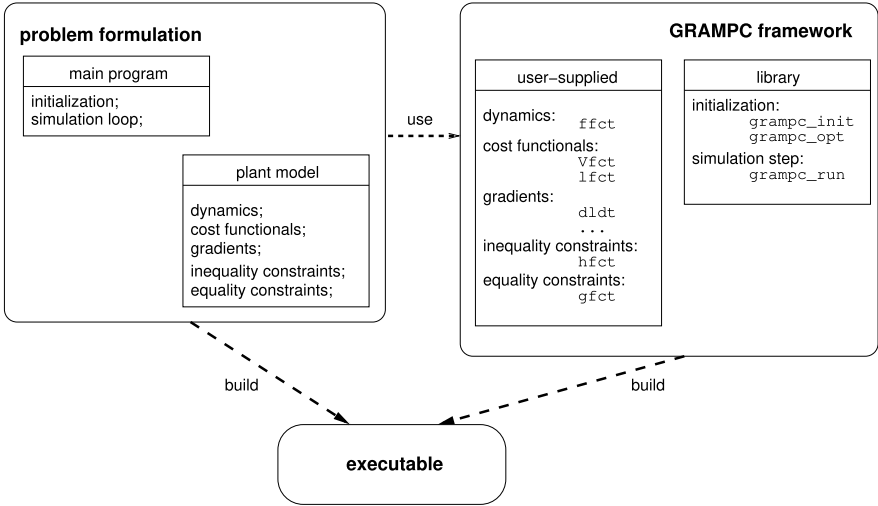


Fig. 1. GRAMPC library schema.

the `grampc` structure should be linked to the buffers where the FMU variables are stored.

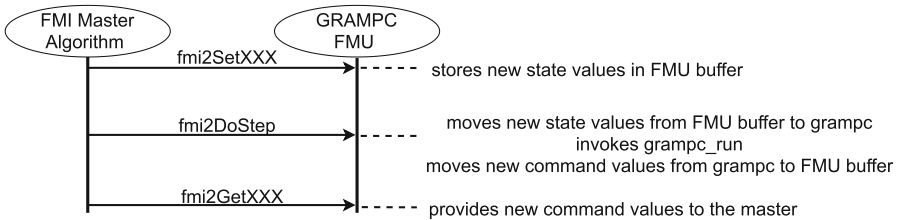


Fig. 2. Interaction between Master and GRAMPC.

#### 4.1 Advantages of GRAMPC as an FMU

The benefits deriving from the proposed approach are related to the general advantages of an FMI based co-simulation, i.e., the possibility of easily coupling the GRAMPC controller with other tools such as Simulink or OpenModelica for the controlled plant component. In particular, the GRAMPC FMU may require a simple model for the predicted trajectory, while a more complex and accurate model can be created for the controlled plant in another FMU, using tools that fit the problem domain. Moreover, the time required to run a co-simulation can be easily reduced by exploiting a simple numerical integration solver (e.g., a Euler solver) in the GRAMPC FMU while a more accurate and computation

demanding solver (e.g., Runge-Kutta) is only used in the plant FMU, which is the one that computes the actual evolution of the system. Thanks to the proposed approach it is also possible to run tests of the GRAMPC controller by exploiting existing features such as the Simulink white noise generator block for sensor errors, or the INTO-CPS Design Space Exploration (DSE) for the analysis of the behaviour with small parametric variations.

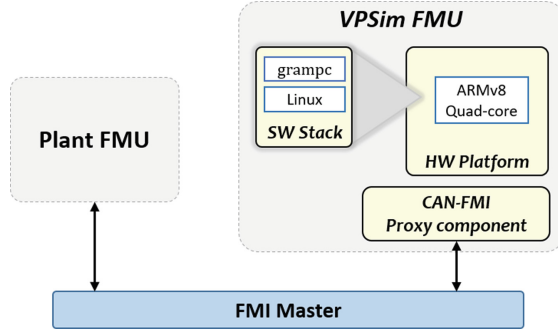
## 4.2 Advantages of Hardware Platform Modelling Within VPSim

Complementary to separating GRAMPC into a standalone FMU, being able to model the `grampc_run` execution on a real hardware is a key to assess performance bottlenecks of the control strategy. This is made possible thanks to the VPSim [7] SoC virtual prototyping capabilities. It was developed with the purpose of accelerating the software/hardware co-validation in the early stages of the design development. VPSim makes it easy to model and emulate various hardware architectures. At the same time, the user can simply test and debug complete software stacks on these emulated architectures. Furthermore, VPSim is distinguished by its ability to host third-party subsystems using many standard and non-standard interfaces. In particular, it fully supports the FMI standard [24]. Therefore, it can interface easily with other modelling tools and simulators within an FMI-based co-simulation. From the user view, FMI in VPSim is exposed as a proxy component that must be connected to a compatible hardware communication interface, such as CAN bus or I2C slave. In addition, VPSim proposes a user-friendly method for automatic generation of the virtual platform FMU, based on a high-level description of the hardware/software platform. Figure 3 shows the general architecture of an FMI co-simulation involving an FMU with a GRAMPC model executed on a processor emulated with VPSim. The deployment of GRAMPC on a simulated architecture with VPSim enables (i) evaluating the behavior of `grampc_run` on the target hardware architecture, (ii) identifying the best hardware support for the control code, and (iii) devising software improvement strategies such as parallel implementation.

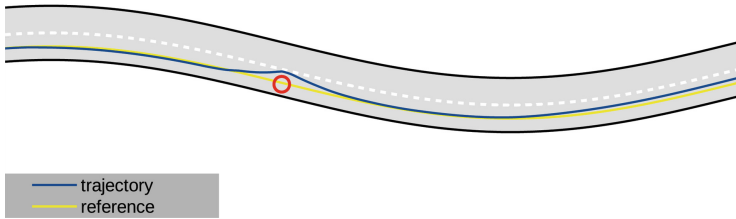
## 5 Case Study

The specific case study concerns the autonomous driving of a vehicle modelled for simplicity by kinematics laws through the GRAMPC library. The problem addressed is to follow a sinusoidal trajectory that follows the carriageway, avoiding some obstacles (modelled as circular areas). Figure 4 shows a possible trajectory of a car avoiding an obstacle (red circle).

The case study is taken from an example available in the GRAMPC distribution. Obviously, in a realistic case, the MPC control algorithm, which is a low-level control, will have to be integrated with the vision and decision system for the waypoints to be reached at each iteration. This information is assumed to be given as input to the system statically, at the beginning of the simulation, and the vision system will not be considered.



**Fig. 3.** The architecture for a co-simulation with VPSim FMU.



**Fig. 4.** Example of a trajectory.

## 5.1 Vehicle Model

The model used for the case study is the kinematic bicycle model of the vehicle shown in Fig. 5, adapted from [23]. This model, commonly used in the field of MPC, approximates a four-wheel vehicle by replacing the two wheels of each axle with one wheel on the longitudinal axis.

The kinematic behaviour of the model is described by the following equations, where the control command inputs are the acceleration  $a$  ( $\text{m/s}^2$ ) and the front wheel steering angle  $\delta$  (rad).

$$\dot{x} = V \cos(\psi + \beta(\delta)) \quad (9)$$

$$\dot{y} = V \sin(\psi + \beta(\delta)) \quad (10)$$

$$\dot{V} = a \quad (11)$$

$$\dot{\psi} = \frac{V}{l_r + l_f} \cos(\beta(\delta)) \tan(\delta) \quad (12)$$

Angle  $\beta$  is the slip angle at the centre of gravity G and it is described by Eq. (13), where  $l_r$  and  $l_f$  are the distances from G of the rear and front wheel, respectively.

$$\beta(\delta) = \arctan\left(\tan(\delta) \frac{l_r}{l_r + l_f}\right) \quad (13)$$



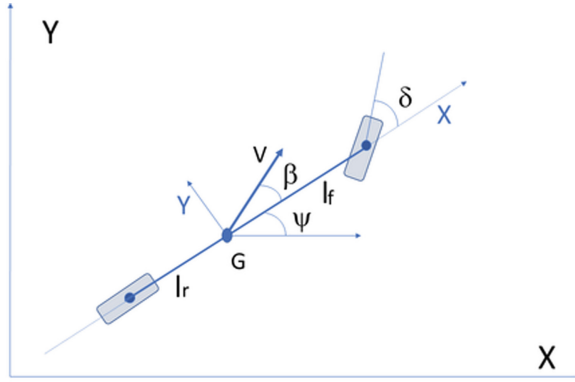


Fig. 5. Kinematic bicycle model of the vehicle, redrawn from [23].

## 5.2 Simulink Model of the Plant

Equations (9), (10), (11), (12), and (13) have been implemented with the Simulink model shown in Fig. 6, using the base blocks of the Simulink trigonometry library. The Simulink model comprises four integrators to output the actual values of the variables. The initial state of these integrators corresponds to the initial values of the plant's state variables.

The Simulink environment generates an FMU whose model parameters (such as  $l_r$ ,  $l_f$ , and the initial state) can be set in the INTO-CPS co-simulation environment. The Simulink environment also chooses the `ode45` variable step size and the default parameters for the explicit Runge-Kutta integrator.

## 5.3 Vehicle and Controller in GRAMPC

The model of the vehicle in GRAMPC is shown in Listing 1.1 and matches the equations shown in Sect. 5.1, using the notation of C. The same model is used to solve the optimisation problem and for executing a self-contained simulation in the framework. Listing 1.2 shows the four optimisation constraints:

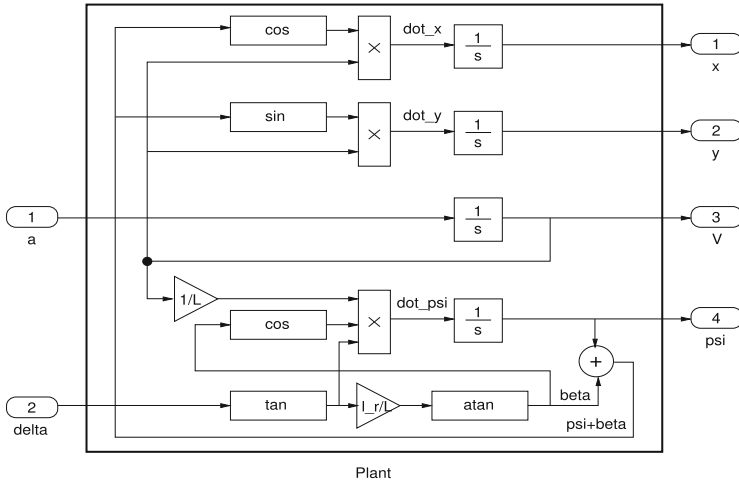
```

1  double beta = ATAN(param[18]*TAN(u[0])/(param[18]+
   param[19]));
2  out[0] = COS(x[2]+beta)*x[3];
3  out[1] = SIN(x[2]+beta)*x[3];
4  out[2] = x[3]*COS(beta)*TAN(u[0])/(param[18]+param
   [19]);
5  out[3] = u[1];

```

Listing 1.1. Implementation of the vehicle model in GRAMPC.

- `out[0]` on line 1 represents the constraint for obstacle avoidance and is represented by a circle of radius 1 located at  $(50, -0.2)$  in the XY plane.



**Fig. 6.** Simulink model of the plant. Parameter  $L$  equals  $l_r + l_f$ .

- `out[1]` on line 2 and `out[2]` on line 3 are the constraints representing the edges of the road, represented by two sinusoids,
- `out[3]` on line 4 represents the speed limit that the vehicle must respect.

```

1  out[0] = (2 - POW2(-50 + x[0]) - POW2(( 0.2 + x[1])))
   );
2  out[1] = -x[1] + 4*SIN(2 * pi * 0.01 * x[0]) - 1.5;
3  out[2] = x[1] - 4*SIN(2 * pi * 0.01 * x[0]) - 4.5;
4  out[3] = x[3] - 40;

```

**Listing 1.2.** Definition of the constraints in GRAMPC.

```

1  State* tick(State* st) {
2  grampc->sol->xnext[0] = (typeRNum)st->x;
3  grampc->sol->xnext[1] = (typeRNum)st->y;
4  grampc->sol->xnext[3] = (typeRNum)st->V;
5  grampc->sol->xnext[2] = (typeRNum)st->psi;
6  grampc_setparam_real_vector(grampc, "x0", grampc->sol->
   xnext);
7  grampc_run(grampc);
8  t = t + grampc->param->dt;
9  st->a = grampc->sol->unext[1];
10 st->delta = grampc->sol->unext[0];
11 }

```

**Listing 1.3.** Algorithm evolution in GRAMPC.

Listing 1.3 shows the custom function `tick`, called by the master every co-simulation step through the `fmi2Dostep` function: lines 3–7 move the values

received from the controlled plant to `grampc`; lines 9–10 invoke the execution of `grampc` and increase the time variable by `dt`; lines 12–13 save the newly generated commands. The co-simulation architecture is shown in Fig. 7, with the INTO-CPS Co-simulation Orchestration Engine (COE) playing the role of the FMI master algorithm.

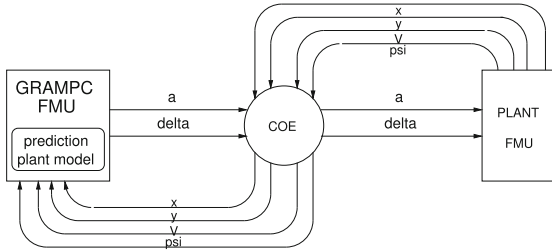


Fig. 7. Co-simulation architecture of the case study.

#### 5.4 Hardware Platform with VPSim

VPSim can simulate a large variety of architectures using both its integrated models and external model providers such as QEMU [1], ARM fast models, or open virtual platforms. In the context of this paper, as shown in Fig. 3, VPSim emulates a quad-core ARMv8 64-bit processor architecture using QEMU. Each core has private L1 & L2 caches. All the cores share four slices of LLC banks, which are connected to the NoC and peripheral devices. The platform runs a Linux OS which executes the GRAMPC algorithm. A CAN controller model provides FMI interfaces and makes it possible to receive and transmit control I/O data to and from the `grampc.run` application that uses the SocketCAN API [25] to retrieve them, as would be the case on real hardware.

It must be stated that a real-time OS could be supported by the proposed methodology. It would be required for industrial development and validation to ensure the periodic scheduling of the `grampc.run` function. Yet, using a standard Linux - executing a single application triggered by CAN events - is relevant for the exploration of the control strategy while accounting for potential execution performance bottlenecks. Indeed, if `grampc.run` executes in less time than the period of CAN messages, it is then periodically executed. Otherwise, it will fail to process all incoming messages and meet its deadlines, as would be the case when considering an RTOS.

#### 5.5 Results

The GRAMPC framework uses a single model as the prediction model, needed for MPC optimization, and as the controlled plant model. While this choice

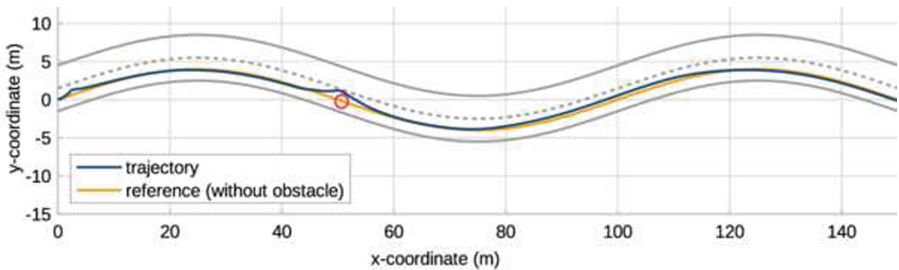
**Table 1.** Parameter values.

Parameter	Value	
$\delta_{\min}$	-0.5	rad
$\delta_{\max}$	+0.5	rad
$a_{\min}$	-11.2	m/s <sup>2</sup>
$a_{\max}$	+5.34	m/s <sup>2</sup>
Front track $l_f$	1.670	m
Rear track $l_r$	1.394	m
Time horizon	1	s
MPC solver	Euler	

is often convenient, it may be the case that an embedded application must use a prediction model simplified with respect to the controlled plant model. In such cases, co-simulation makes it possible to use two distinct models. As a preliminary step towards the co-simulation of distinct models, in this work it has been checked that co-simulation does not introduce significant deviations from the case of GRAMPC simulation with a single model. In order to verify the consistency between the two simulation methods, the same mathematical vehicle model has been implemented in C for the prediction model, and in Simulink for the plant model. The results of co-simulations are consistent with the results of self-contained simulations in GRAMPC, producing a difference less than 1 mm.

This section reports results of co-simulations in case of the decoupling of the two models, making the co-simulated system more realistic with respect to the GRAMPC self-contained one. The main parameters of the analyzed scenarios are shown in Table 1.

**Nominal Co-simulation Results.** The vehicle starts at the position (0,0) and must follow the sinusoidal trajectory avoiding the obstacle at (50, -0.2). Figure 8 shows a run with a fixed step size of 0.001 s and an end time of 20 s.

**Fig. 8.** Results of a co-simulation run.

The maximum computation time of the GRAMPC algorithm is less than a millisecond and the simulation time is 50s on an Intel<sup>®</sup> Core<sup>™</sup> i7-7700 CPU @ 3.60 GHz  $\times$  8. The choice of the Euler solver inside the MPC model, together with the GRAMPC setup, leads to a computation time less than the co-simulation step-size, which guarantees a realtime-like throughput. Notoriously, the Euler solver is computationally less demanding than other solvers.

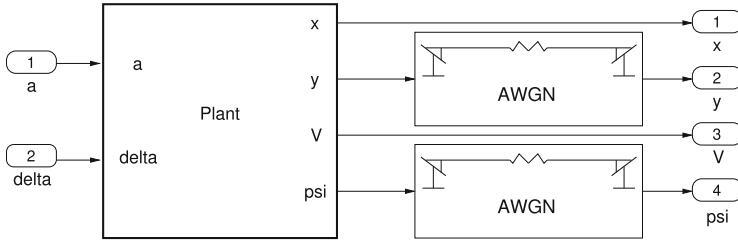
As shown in Fig. 8, the vehicle follows the trajectory avoiding the obstacle. The mean error of the actual trajectory against the reference trajectory without the obstacle is 0.08 m (first row in Table 2) and the maximum absolute error is 0.76 m, both evaluated excluding the area around the obstacle. With respect to the limits imposed on the constrained optimisation problem, as far as the obstacle avoidance section is concerned, it was verified that the trajectory calculated by GRAMPC is such that the centre of mass of the vehicle completely avoids the obstacle. This translates into verifying that the distance between the trajectory and the centre of the obstacle is always greater than the radius of the circle that formally defines the obstacle region itself. The closest distance between the obstacle and the centre of mass of the vehicle is 0.4 m.

**Response to Physical Parameter Variation.** By exploiting the decoupling of the model used within GRAMPC and the model used for plant in Simulink it is possible to run robustness tests against small variations of the physical parameters of the vehicle under analysis. Table 2 shows the four different scenarios where the parameters  $l_f$  and  $l_r$  of the Simulink model have a  $\pm 5\%$  deviation with respect to the nominal values used in the first experiment, reported in the first row of the table. This variation of the parameters emulates a reasonable measurement error. It may be stated that the GRAMPC algorithm is robust as the mean error is scarcely affected by physical changes. Moreover, the maximum absolute error is not affected by physical variations and therefore it is not shown.

**Table 2.** Different parameters.

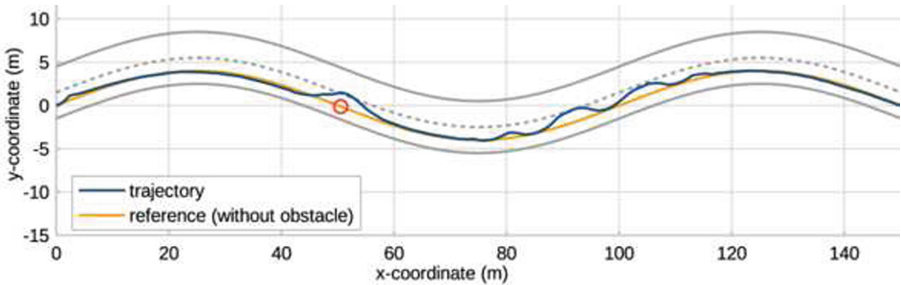
Front track $l_l$ (m)	Rear track $l_r$ (m)	Error (m)
1.67	1.394	0.08
1.67	1.464	0.09
1.67	1.324	0.09
1.75	1.394	0.08
1.59	1.394	0.10

**Enabling Perturbation Analysis.** Figure 10 shows the results of the co-simulation with a perturbation in the  $y$  and  $\psi$  values produced by the plant FMU and consumed by the GRAMPC algorithm. The perturbation has been



**Fig. 9.** Simulink model of the plant with AWGN.

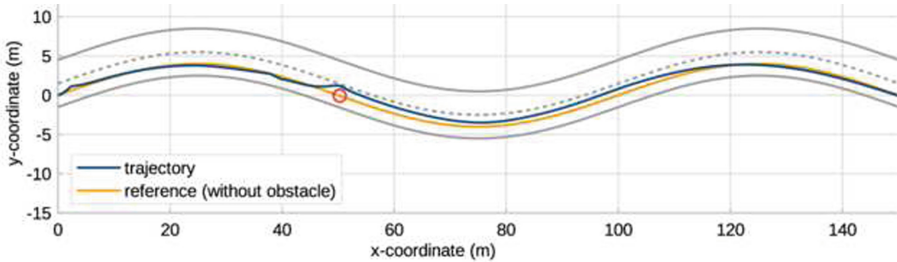
implemented with the Simulink AWGN block, which has been applied to the  $y$  coordinate with a variance of 0.1 (10 cm of measurement error) and to angle  $\psi$  with a variance of 0.01 ( $1^\circ$  of measurement error), obtaining the model in Fig. 9. As shown in Fig. 10, the vehicle is still capable of avoiding the obstacle but the error has increased to 0.240 m and the maximum absolute error has increased to 1.58 m. The framework can be used for perturbation analysis by considering more cases.



**Fig. 10.** Results of a co-simulation run with AWGN on sensors.

**Co-simulation with VPSim.** As a benefit of the proposed approach, it is possible to generate the FMU that executes the `grampc_run` algorithm on a specific hardware platform. In the following, we show the nominal co-simulation in case of GRAMPC executed on top of an ARMv8 quad-core processor emulated with VPSim. With respect to Fig. 7, the FMU generated with VPSim can replace the GRAMPC FMU. The results, shown in Fig. 11, present a mean error of 0.262 m, while the vehicle is still able to avoid the obstacle. This increase in the mean error is consistent with the fact that the average execution time of `grampc_run` (2.4 ms) is longer than the expected co-simulation stepsize (1 ms). From this point on, several improvement strategies to the `grampc_run` implementation could be sought by the designer such as parallelizing the algorithm,

changing the prediction window or the optimization solver, or even choosing more appropriate hardware.



**Fig. 11.** Results of a co-simulation run with VPSim.

## 6 Conclusions

This paper has shown an approach to enable the analysis of an MPC algorithm through co-simulations involving relevant aspects of three different domains: (i) physical laws, defining the evolution of the system, (ii) control algorithm, optimising the response of the system, and (iii) processor architecture imposing constraints on the execution time. The proposed approach is based on the strategy of embedding into an FMU a GRAMPC controller running on a VPSim-simulated processor to assess the performance of the processor.

The case study has shown some of the possibilities opened by the proposed approach, such as the response to physical parameter variations or the evaluation of the impact of processor architecture on the system. Each different analysis can be extended with knowledge and tools deriving from the respective domain. Users expert in parallel computation could improve the architecture performances by optimising the code, users expert in fault tolerance could decrease the impact of faulty sensors by applying, for example, redundancy in the plant model. Finally, experts in MPC could be interested in finding the best trade-off between accuracy and performance. Working all together on the same artefacts, and combining effort in different fields would lead towards the implementation of optimal and robust systems.

As further work, in order to get better results in the parallelisation of MPC, it could be also interesting to investigate the usage of a quite different structure of the control algorithm allowing a more effective parallelism.

## References

1. Bellard, F.: QEMU, a fast and portable dynamic translator. In: 2005 USENIX Annual Technical Conference (USENIX ATC 05). USENIX Association, Anaheim, CA (2005)

2. Bernardeschi, C., et al.: Cross-level co-simulation and verification of an automatic transmission control on embedded processor. In: Cleophas, L., Massink, M. (eds.) *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops*. LNCS, vol. 12524, pp. 263–279. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67220-1\\_20](https://doi.org/10.1007/978-3-030-67220-1_20)
3. Bernardeschi, C., Dini, P., Domenici, A., Palmieri, M., Saponara, S.: Formal verification and co-simulation in the design of a synchronous motor control algorithm. *Energies* **13**(16), 4057 (2020). <https://doi.org/10.3390/en13164057>
4. Bernardeschi, C., Domenici, A., Masci, P.: A PVS-simulink integrated environment for model-based analysis of cyber-physical systems. *IEEE Trans. Soft. Eng.* **44**(6), 512–533 (2018). <https://doi.org/10.1109/TSE.2017.2694423>
5. Blochwitz, T., et al.: Functional mockup interface 2.0: the standard for tool independent exchange of simulation models. In: *Proceedings of the 9th International MODELICA Conference*, pp. 173–184, no. 76 in *Linköping Electronic Conference Proceedings*, Linköping University Electronic Press (2012). <https://doi.org/10.3384/ecp12076173>
6. Ceusters, G., et al.: Model-predictive control and reinforcement learning in multi-energy system case studies. In: *eprint 2104.09785* (2021)
7. Charif, A., Busnot, G., Mameesh, R.H., Sassolas, T., Ventroux, N.: Fast virtual prototyping for embedded computing systems design and exploration. In: Chillet, D. (ed.) *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO 2019, Valencia, 2019*, pp. 3:1–3:8. ACM (2019). <https://doi.org/10.1145/3300189.3300192>
8. Dini, P., Saponara, S.: Design of adaptive controller exploiting learning concepts applied to a BLDC-based drive system. *Energies* **13**(10), 2512 (2020). <https://doi.org/10.3390/en13102512>
9. Domenici, A., Fagiolini, A., Palmieri, M.: Integrated simulation and formal verification of a simple autonomous vehicle. In: Cerone, A., Roveri, M. (eds.) *Software Engineering and Formal Methods*. LNCS, vol. 10729, pp. 300–314. Springer Int. Publishing, Cham (2018). [https://doi.org/10.1007/978-3-319-74781-1\\_21](https://doi.org/10.1007/978-3-319-74781-1_21)
10. Englert, T., Völz, A., Mesmer, F., Rhein, S., Graiche, K.: A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (GRAMPC). *Optim. Eng.* **20**, 769–809 (2019). <https://doi.org/10.1007/s11081-018-9417-2>
11. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: a survey. *ACM Comput. Surv. (CSUR)* **51**(3), 1–33 (2018). <https://doi.org/10.1145/3179993>
12. Gräber, M., Kirches, C., Scharff, D., Tegethoff, W.: Using functional mock-up units for nonlinear model predictive control. In: *9th International MODELICA Conference vol. 076*, pp. 781–790 (2012). <https://doi.org/10.3384/ecp12076781>
13. Grüne, L., Pannek, J.: *Nonlinear Model Predictive Control - Theory and Algorithms*. Springer, London (2017). <https://doi.org/10.1007/978-0-85729-501-9>
14. Guiggiani, M.: *The Science of Vehicle Dynamics - Handling, Braking, and Ride of Road and Race Cars*. Springer, Dordrecht (2018). <https://doi.org/10.1007/978-94-017-8533-4>
15. He, W., Gao, H., Zhou, C., Yang, C., Li, Z.: Reinforcement learning control of a flexible two-link manipulator: an experimental investigation. *IEEE Trans. Syst. Man Cybern. Syst.* **51**(12), 7326–7336 (2020). <https://doi.org/10.1109/TSMC.2020.2975232>



16. Larsen, P.G., et al.: Integrated tool chain for model-based design of cyber-physical systems: the INTO-CPS project. In: *Modelling, Analysis, and Control of Complex CPS (CPS Data)*, 2016 2nd International Workshop on, pp. 1–6. IEEE (2016)
17. Lee, D., Lim, M.C., Negash, L., Choi, H.L.: EPPY based building co-simulation for model predictive control of HVAC optimization. In: *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, pp. 1051–1055 (2018)
18. Legaard, C., Tola, D., Schranz, T., Macedo, H., Larsen, P.: A universal mechanism for implementing functional mock-up units. In: *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - SIMULTECH*, pp. 121–129. SciTePress (2021)
19. Lenord, O., et al.: eFMI: an open standard for physical models in embedded software. In: *Proceedings of the 14th International Modelica Conference 2021*, pp. 57–71. Modelica Association (2021). <https://doi.org/10.3384/ecp2118157>
20. Palmieri, M., Bernardeschi, C., Masci, P.: Co-simulation of semi-autonomous systems: the Line Follower Robot case study. In: Cerone, A., Roveri, M. (eds.) *Software Engineering and Formal Methods*. LNCS, vol. 10729, pp. 423–437. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-74781-1\\_29](https://doi.org/10.1007/978-3-319-74781-1_29)
21. Palmieri, M., Bernardeschi, C., Masci, P.: A framework for FMI-based co-simulation of human-machine interfaces. *Softw. Syst. Model.* **19**(3), 601–623 (2020). <https://doi.org/10.1007/s10270-019-00754-9>
22. Palmieri, M., Macedo, H.D.: Automatic generation of functional mock-up units from formal specifications. In: Camara, J., Steffen, M. (eds.) *Software Engineering and Formal Methods*, pp. 27–33. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-57506-9\\_3](https://doi.org/10.1007/978-3-030-57506-9_3)
23. Polack, P., Althé, F., Novel, B., de La Fortelle, A.: The kinematic bicycle model: a consistent model for planning feasible trajectories for autonomous vehicles? In: *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 812–818 (2017). <https://doi.org/10.1109/IVS.2017.7995816>
24. Saidi, S.E., Charif, A., Sassolas, T., Guay, P.G., Souza, H., Ventroux, N.: Fast virtual prototyping of cyber-physical systems using SystemC and FMI: ADAS use case. In: *Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP 2019)*, pp. 43–49. Association for Computing Machinery, USA (2019). <https://doi.org/10.1145/3339985.3358488>
25. SocketCAN (2020). <https://www.kernel.org/doc/html/latest/networking/can.html>
26. Ventroux, N., et al.: SESAM: an MPSoC simulation environment for dynamic application processing. In: *2010 10th IEEE International Conference on Computer and Information Technology*, pp. 1880–1886 (2010). <https://doi.org/10.1109/CIT.2010.322>
27. Von Wissel, D., Talon, V., Thomas, V., Grangier, B., Lansky, L., Uchanski, M.: Linking model predictive control (MPC) and system simulation tools to support automotive system architecture choices. In: *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, France (2016)
28. Yurtsever, E., Lambert, J., Carballo, A., Takeda, K.: A survey of autonomous driving: common practices and emerging technologies. *IEEE Access* **8**, 58443–58469 (2020). <https://doi.org/10.1109/ACCESS.2020.2983149>