Chattering-Free Simulation of Hybrid Dynamical Systems with the Functional Mock-Up Interface 2.0

Ayman Aljarbouh[1] Benoit Caillaud[1]

[1]Centre de Recherche INRIA-IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France,
{ayman.aljarbouh,benoit.caillaud}@inria.fr

Abstract

The numerical simulation of non-smooth hybrid systems exhibiting chattering behavior requires high computational costs. In the worst case, the simulation appears to come to a halt, since infinitely many discrete transitions would need to be simulated. In this paper we present an FMI-based framework and prototypical implementation for robust and reliable detection and elimination "On the Fly" of chattering behavior in run-time simulation of non-smooth hybrid systems. The main benefit of the developed framework is that it establishes solvability requirements and theorems for simulating hybrid systems while performing the chattering path avoidance internally in the master algorithm of the interface. This increases the efficiency of the chattering-free simulation as no enumeration of modes is required during the chattering detection and elimination process. The developed FMI-based framework can generate a chattering-free simulation for any generic chattering Functional Mockup Unit (FMU) conforming to the FMI standard v2.0 Specification for model exchange.

Keywords: Functional Mockup Interface (FMI), Functional Mockup Unit (FMU), Non-smooth Hybrid systems, Discontinuity mappings, Chattering

1 Introduction

In the literature, the term "hybrid systems" is used to describe a very wide class of dynamical systems with interacting continuous and discrete dynamics. The state variables in such systems are capable of evolving continuously (flowing) and/or evolving discontinuously (jumping). That is, the presence of two different behaviors, continuous and discrete, is the cause of heterogeneity (Zhang et al., 2001; Cai et al., 2008). However, even simple hybrid systems can exhibit many unique phenomena, such as *chattering behavior*. The interaction between time-driven continuous variable dynamics (i.e. ODEs, DAEs) and event-driven discrete logic dynamics (i.e. `If-then-else`) may lead to this non-smooth behavior, which can be intuitively thought of as involving infinitely fast and continuous switching between different control actions or modes of operation (Aljarbouh and Caillaud, 2015b). Models of physical hybrid systems may be chattering due to modeling over-abstraction, actuators limitations, time discretization, or unmodeled dynamics (usually from servomechanisms, sensors and data processors with small time constants).

1.1 Problem Statement

As in physical hybrid systems there is no chattering, it is not reasonable then to assume that the control signal time evolution can chatter or switch at infinite frequency. This undesirable significant oscillation with an infinitely fast frequency components of the control propagate through the system, because of chattering, affects the system output. In particular, chattering control is harmful because it leads to low control accuracy, and once applied, can lead to high wear of moving mechanical parts, as well as high heat losses in electrical power circuits. In addition, the numerical simulation of hybrid systems exhibiting chattering behavior requires high computational costs as small step-sizes are required to maintain the numerical precision. For both non-adaptive and adaptive time stepping with event localization, root finding to locate the exact time of occurrence of the chattering event causes continuous integration to become dramatically and excessively slow. The system converges fast to the point in time at which infinitely many discrete transitions need to be simulated, and the simulation then appears to come to a halt. Chattering behavior has to be treated in an appropriate way to ensure that the numerical integration progress terminates in a reasonable time. This has been investigated by means of different methods. A smooth sliding motion can be induced on the switching manifold on which the chattering occurs (Leine and Nijmeijer, 2004; di Bernardo et al., 2008; Biák et al., 2013; Weiss et al., 2015). Filippov Differential inclusion approach (Filippov, 1988) can be used in this case to define equivalent sliding dynamics on the switching manifold on which the chattering occurs. Another approach (the so-called equivalent control) proposed by Utkin (Utkin,

1992) can also be used. However, the computation of the equivalent dynamics turns to be difficult whenever the system chatters between more than two dynamics. This arises when the chattering behavior occurs in dynamical systems having multiple discontinuous control variables. In the Functional Mock-up Interface (FMI) specification, Functional Mockup Units (FMUs) should add a small hysteresis to the event indicators to avoid chattering (Blochwitz et al., 2012). This approach has the following disadvantages: I) A Modelica tool will also add a hysterisis when handling state events, to ensure that the zero crossings happen with non-zero values of the input arguments of the event functions at the integration restart. Therefore, when calling the FMI function `fmi2GetEventIndicators` from the Modelica model, it will introduce the hysteresis twice to the event indicators, and as a result, the resulting event triggered by the imported FMU is slightly inaccurate. II) Adding hysterisis to the event indicators does not guarantee an efficient treatement of the chattering behavior, as the physics in chattering hybrid systems make the solution $x_\varepsilon(\cdot)$ be a saw-toothed, or zigzag function, i.e., a function that oscillates around the switching surface, with peaks at $-\varepsilon < 0$ and $+\varepsilon > 0$, with $t_{i+1} - t_i = 2\varepsilon$ (see Example 1 in Section 2.3). III) The size of the small value ε shall be related to the size of the event indicator z_j. The interface then would become more complicated, because, in order to determine the size of ε in the simulation environment which imports an FMU, the "nominal" value of z_j has to be reported by the FMU, which requires more information from the tool that generated the FMU, but cannot be handled efficiently in the simulation environment that calls the FMU. IV) If this would be handled in the simulation environment, there is always the danger that the environment does not handle it properly, but the FMU would be blamed for a failure.

1.2 Contribution

In this paper, we present methods and techniques for treating chattering behavior of non-smooth hybrid dynamical systems in the context of the Functional Mock-up Interface (FMI), and a prototypical implementation. In particular we discuss technical issues and implementation of a generic FMI which rigorously detects and eliminates chattering behavior in run-time simulation without modes enumeration, and without any need to add a small hysteresis to the event indicators in the FMUs. The developed chattering-free FMI localizes the non-smooth structural changes in the system in an accurate way and allows sliding mode simulation when the chattering occurs. It treats the chattering non-smoothness in the trajectory of the state variables by a smooth correction after each integration time-step. Furthermore, our chattering-free FMI can robustly handle the case of chattering on the intersection of finitely many switching manifolds iteratively without any need to solve stiff nonlinear equations for the computation of the chattering-free coefficients. In addition, this paper provides a guidance for development of a hybrid chattering-free version of the Functional Mockup Interface (FMI) standard, giving a computational framework for an ideal manipulation of chattering behavior.

The paper is organized as follows: Section 2 gives a closer look into how the chattering behavior occurs in hybrid systems, as well as the challenges when simulating hybrid systems with chattering executions. Afterwards, we present in Section 3 the chattering-free semantics for reliable detection and elimination of chattering behavior in run time simulation. In Section 4, a prototype implementation is sketched for applying the chattering-free computational framework from Section 3 to the Functional Mock-Up Interface v2.0 for Model Exachange. Finally, the simulation results and conclusions of the work are given in Section 5 and Section 6 respectively. We illustrate the concepts with examples throughout the paper.

2 Chattering in Hybrid Systems

Formally we define chattering executions as solutions to hybrid systems having infinitely many discrete transitions in finite time. This happens when nearly equal thresholds for the transitions conditions of different modes are satisfied and the system start to oscillate around them. Numerical errors may also be the source of chattering as transitions conditions can be satisfied because of local errors. In chattering behavior, the system moves back and forth between modes, that is, the gradient of continuous-time behavior in each one of two adjacent modes is directed towards their common switching surface. When in either of the two adjacent modes on the common switching surface, an infinitesimal step causes a mode change. In the new mode, the gradient directs behavior to the previous mode and after another infinitesimal step a change to the previous mode occurs.

2.1 Chattering Execution

An *execution* χ of a hybrid system is chattering if there exist finite constants τ_∞ and C such that

$$\lim_{i\to\infty} \tau_i = \sum_{i=0}^{\infty} (\tau_{i+1} - \tau_i) = \tau_\infty \tag{1}$$

$$\forall i \geq C: \quad \tau_{i+1} - \tau_i = 0 \tag{2}$$

where $\{\tau_i\}_{i\in\mathbb{N}}$ is a set of strictly increasing time instants represents discontinuity points (state events instants).

2.2 Chattering and Simulation

An essential element of numerical simulation of a hybrid dynamical system is the generation of discrete events

DOI 10.3384/ecp1612495

from continuous variables that exceed thresholds. Generating these events is generally implemented using relational operations (e.g. >, >=, <, <=). For an accurate simulation, the point in time at which these relations change their truth value has to be located within a small tolerance. A zero-crossing function $g(t,x)$ can be used to identify the boundary at which the change takes place. Usually state variables x are used as argument to the event indicator $g(t,x)$. The nature of zero-crossing detection and location is to compare the sign of the function value $g(t,x)$ at the beginning and the end of each time integration step, and if it changes, declare that it crossed zero and then bracketing the zero-crossing event (i.e. bisectional search) to locate the zero-crossing. During the search process, the values of state variables x, needed for the computation of $g(t,x)$, are evaluated by interpolation, using the values $x(t_i)$ and $x(t_{i+1})$. Because of the nature of finite precision arithmetic on digital computers, the time that the event occurred can only be located within an interval $[T_{left}, T_{right}]$ that corresponds to machine precision. During each iteration of the zero-crossing location, the zero-crossing function is evaluated twice: at the left and the right side of the reducing interval. After the event is bracketed by T_{left} and T_{right}, the ODE solver first advances integration time from t_i to T_{left}. The solver is then reset before advancing to T_{right} followed by switching the mode. In doing so, the assumption of continuity holds throughout the numerical integration. However, this approach may fail if the system exhibits a chattering execution. The zero crossing function $g(t,x)$ is a function of the model state, but it does not contribute to its continuous dynamics $f(t,x)$. Therefore, the numerical integration can proceed without taking the dynamics of $g(t,x)$ into account, and when these are faster than the dynamics $f(t,x)$, the chattering execution then causes the previous T_{right} to become the T_{left} of the next time step, and the integration will move with the minimum step size allowed. In order to illustrate the simulation of a chattering execution, a simple example shall be given.

2.3 Example 1: Relay Feedback

The relay feedback system is a good candidate to show the chattering behavior of a hybrid dynamical system (Aljarbouh and Caillaud, 2015a). The relay feedback system consists of a dynamical system and a sign function connected in feedback. The sign function leads to a discontinuous differential equation (Johansson et al., 2002). Consider the following example for $x = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3}$$
$$y(t) = Cx(t) \tag{4}$$
$$u(t) = -sgn(y(t)) \tag{5}$$

$$A = \begin{bmatrix} -3 & 1 & 0 \\ -3 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ -2\beta \\ \beta^2 \end{bmatrix} \tag{6}$$
$$C = [1 \ 0 \ 0] \tag{7}$$

The system in this example is represented as a hybrid system with two control modes q_1 and q_2 where the phase space of the system is split by a single switching manifold $\Gamma = \{x \in \mathbb{R}^n : \ g(t,x) = 0\}$ into two domain: $D_1 = \{x \in \mathbb{R}^n : \ g(t,x) < 0\}$ and $D_2 = \{x \in \mathbb{R}^n : \ g(t,x) > 0\}$ so that opposed zero crossing of the switching function $g(t,x) = x_1(t)$ defines the switching from q_1 to q_2 and vice-versa (e.g. a switching from D_1 to D_2 occurs when $g(t,x)$ changes its domain to $g(t,x) \geq 0$). It is important to recognize that the "zero crossing" approach defined by available integrators, for detecting state events, requires that the event function variables are non-zero at the event instant and after initialization. So, suppose one integrates the differential equation 3 with some delay in the control switch between $+1$ and -1 because some kind of hysteresis function implemented around the switching surface $x_1 = 0$. In addition to use it for handling the non-zero domain change of event functions, such a procedure is sometimes used in order to avoid too many switches. Even with adding such hysterisis, the physics in this system, because of chattering, makes the solution $x_{1\varepsilon}(\cdot)$ to be a saw-toothed, or zigzag function, i.e., a function that oscillates around $x_1 = 0$, with peaks at $-\varepsilon < 0$ and $+\varepsilon > 0$, where $t_{i+1} - t_i = 2\varepsilon$. Let the hysteresis size go to zero, i.e., $\varepsilon \to 0$. Then $x_{1\varepsilon}(\cdot)$ converges uniformly towards the zero function. Clearly the number of "events" goes to infinity on any interval of time with positive measure.

Hybrid systems simulation tools struggle even with such naive chattering hybrid system. For example, consider OpenModelica, and Acumen. In OpenModelica, for a data set $\beta = 0.5$ and $x_0 = [0.5 \ 3 \ 0.1]^T$, the solver gets stuck and the simulation terminates with a halt when the execution of the hybrid system start to exhibit a chattering. OpenModelica reports the following error message: *Chattering detected around time 1.88743591101..1.88743593454 (100 state events in a row with a total time delta less than the step size).*

```
model Example1
parameter Real x10 = 0.5;
parameter Real x20 = 3.0;
parameter Real x30 = 0.1;
Real x1,x2,x3,u;
initial equation
x1 = x10;
x2 = x20;
x3 = x30;
equation
der(x1) = -3 * x1 + x2 + u;
der(x2) = -3 * x1 + x3 - u;
der(x3) = -x1 + 0.25 * u;
when x1 < 0 then
u = 1;
```

```
elsewhen x1 > 0 then
u = -1;
end when;
end Example1;
```

Acumen language was developed as an extension of event-driven formalisms that have a similar flavor to synchronous languages. In Acumen, models are simulated by a fixed time stepping with fine interleaving of a sequences that can consist of multiple discrete computations followed by a single computation updating the values that should evolve continuously (i.e. global fixed point semantics). Thus, simulating what is happening at any single instance in time consists of zero or more discrete steps followed by a single continuous step. The Acumen model of the system in Example 1 can be written as following:

```
model Main(simulator) =
initially
x1 = 0.5, x2 = 3, x3 = 0.1,
x1'= 0,   x2'= 0, x3'= 0, u = 0

always
x1'= -3*x1 + x2 + u,
x2'= -3*x1 + x3 - u,
x3'= -x1 + 0.25*u,
if x1 >= 0 then u= -1
elseif x1 <= 0 then u= 1 noelse,
simulator.timeStep += 0.0001,
simulator.endTime  += 10
```

Figure 1 shows the fixed time step simulation of Example 1 in Acumen language without events localization. With a fixed step size of 0.0001, the solution trajectory exhibits an undesirable oscillations around the switching surface Γ, with high frequency components of the control switching propagate through the system.

3 Detection and Elimination of Chattering

We consider a hybrid system $\mathscr{H}$ with a finite set of discrete states $q \in Q$ with transverse invariants (Lygeros et al., 2008), where the state space is split into different regions (invariants) $D_q \in \mathbb{R}^n$ by the intersection of p transversally intersected $\mathbb{R}^{n-1}$ switching manifolds Γ_j defined as the zeros of a set of scalar functions $g_j(t,x)$ for $j = 1,2,...,p$,

$$\Gamma_j = \{x \in \mathbb{R}^n : \; g_j(t,x) = 0 \; ; \quad j = 1,2,...,p\} \tag{8}$$

All switching functions $g_j(t,x)$ are assumed to be analytic in their second arguments, (i.e. $\frac{\partial g_j(t,x)}{\partial x} \neq 0$), so that, for each one of the intersected switching manifolds Γ_j, the normal unit vector $\perp_j = \frac{\frac{\partial g_j(t,x)}{\partial x}}{||\frac{\partial g_j(t,x)}{\partial x}||}$, orthogonal to the tangential plane $T_x(\Gamma_j)$, is well defined. Moreover, the normal unit vectors are linearly independent for all the $\mathbb{R}^{(n-r)}$ swicthing intersections where $r \in \{2,3,...,n\}$. The flow map vector field $f(t,x)$ of the hybrid system is discontinuous on all the switching surfaces Γ_j. Therefore, we can associate to each discontinuity surface Γ_j a discontinuous vector field of the form:

$$\dot{x} = f_j(t,x) = \left\{ \begin{array}{l} f_{j1}(t,x) \;\; for \;\; x \in D_{j1} \\ f_{j2}(t,x) \;\; for \;\; x \in D_{j2} \end{array} \right\} \tag{9}$$

$$D_{j1} = \{x \in \mathbb{R}^n : g_j(t,x) < 0\} \tag{10}$$

$$D_{j2} = \{x \in \mathbb{R}^n : g_j(t,x) > 0\} \tag{11}$$

so that opposed zero crossing of $g_j(t,x)$, defines the switching from D_{j1} to D_{j2} and vice versa. Note that, equation 9 represents the necessary condition for the hybrid system to accept a chattering execution between D_{j1} and D_{j2}. If this necessary condition is satisfied for all $j = \{1,2,\cdots,k\}$ with $k \leq p$, then the hybrid system is said to accept a chattering on switching intersection. As each discontinuity surface Γ_j splits the phase domain into two different invariants $D_{j1} \in \mathbb{R}^n$ and $D_{j2} \in \mathbb{R}^n$, the entire continuous domain of the hybrid system $\mathscr{H}$ is then partitioned into 2^p open convex regions $D_q \in \mathbb{R}^n$, in which the solution trajectory flow is governed by the dynamics $f_q(t,x)$, where $q = 1,\cdots,2^p$, and p is the total number of the intersected switching manifolds Γ_j. It is assumed that f_q are smooth in the state x for all D_q. For more details on how the chattering occurs on switching intersection, we refer the reader to (Aljarbouh and Caillaud, 2015b).

3.1 Chattering Detection

Upon crossing a switching manifold Γ_j, the behavior of the solution trajectory can uniquely be characterized by the gradients of the continuous-time behavior according to the dynamics f_{j1} and f_{j2} in a small neighborhood on the both sides of Γ_j. This is given by the normal projections of the dynamics f_{j1} and f_{j2} onto Γ_j (i.e. directional derivatives or Lie derivatives $\mathscr{L}_f g_j(t,x)$), given by

$$f_{j1}^{\perp_j}(t,x) = \mathscr{L}_{f_{j1}} g_j(t,x) = \left(\frac{\partial g_j(t,x)}{\partial x}\right) \cdot f_{j1}(t,x) \tag{12}$$

$$f_{j2}^{\perp_j}(t,x) = \mathscr{L}_{f_{j2}} g_j(t,x) = \left(\frac{\partial g_j(t,x)}{\partial x}\right) \cdot f_{j2}(t,x) \tag{13}$$

The sufficient condition for the hybrid system $\mathscr{H}$ to exhibit a chattering back and forth between the two domains D_{j1} and D_{j2} requires that the necessary condition of chattering is satisfied (equation 9), as well as the following two constraints:

1. a zero crossing on Γ_j (state event) is detected in the integration time interval $[t_i, t_{i+1}]$, that is,

$$g_j(t_i,x_i) \cdot g_j(t_{i+1},x_{i+1}) < 0 \tag{14}$$

2. the scalar inner porduct of the normal projections $f_{j1}^{\perp_j}(t_i,x_i)$ and $f_{j2}^{\perp_j}(t_{i+1},x_{i+1})$ is strictly negative,

DOI 10.3384/ecp1612495

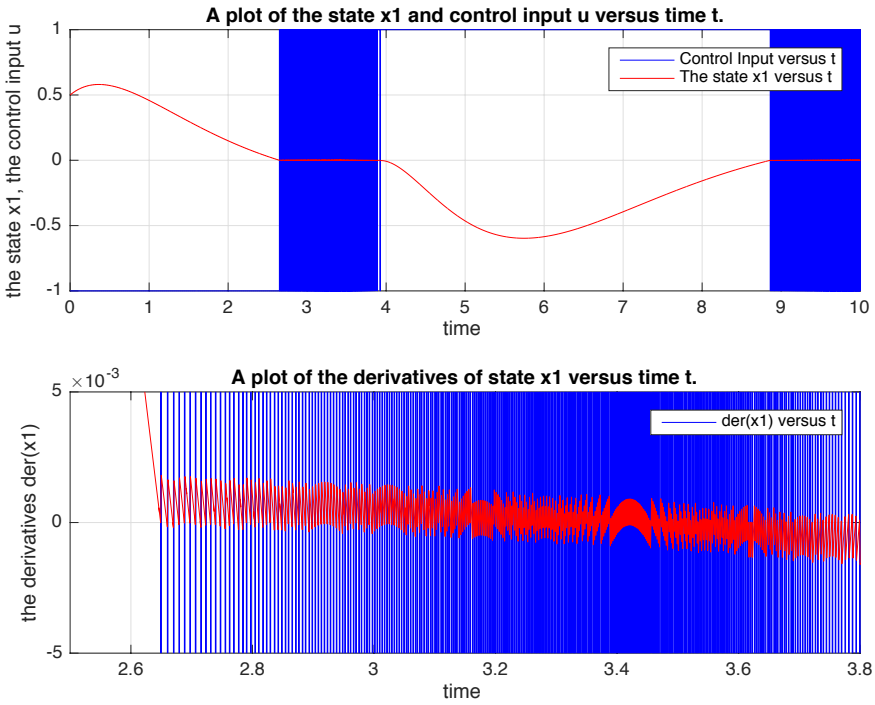

Figure 1. Fixed time step simulation of Example 1 in Acumen without event localization for $\beta = 0.5$ and $x_0 = [0.5\ 3\ 0.1]^T$: Up: time evolution of the event function and the control input with high chattering oscillation. Down: zoom on the first chattering window around the switching surface $x_1(t) = 0$.

that is, the projctions of the two different dynamics (normal onto Γ_j), before and after the zero-crossing, have opposed signs (Figure 2),

$$f_{j1}^{\perp_j}(t_i,x_i)\cdot f_{j2}^{\perp_j}(t_{i+1},x_{i+1}) < 0 \qquad (15)$$

A chattering takes place on the intersection of $k \in \mathbb{N}$ switching manifolds Γ_j if the necessary condition of chattering is satisfied, and for all $j = 1,2,\cdots,k \leq p$, the following three constraints are satisfied:

$$g_j(t_i,x_i)\cdot g_j(t_{i+1},x_{i+1}) < 0 \qquad (16)$$

$$g_j(t_{i+1}(\sigma)) = \kappa;\quad \sigma \in (0,1);\quad \kappa \in (-\varepsilon,\varepsilon) \qquad (17)$$

$$f_{j1}^{\perp_j}(t_i,x_i)\cdot f_{j2}^{\perp_j}(t_{i+1},x_{i+1}) < 0 \qquad (18)$$

In this case of chattering on the intersection of finitely many switching manifolds, the excution of the hybrid system $\mathscr{H}$ chatters back and forth between all the domains D_q in the neighborhood of the intersection.

3.2 Chattering Elimination

One way to prevent the chattering is to keep the solution trajectory in a sliding motion on the switching manifold/intersection on which the chattering occurs. An additional mode, sliding mode, can be inserted into the system to represent the equivalent chattering-free dynamics. For all the switching manifolds Γ_j, the dynamics

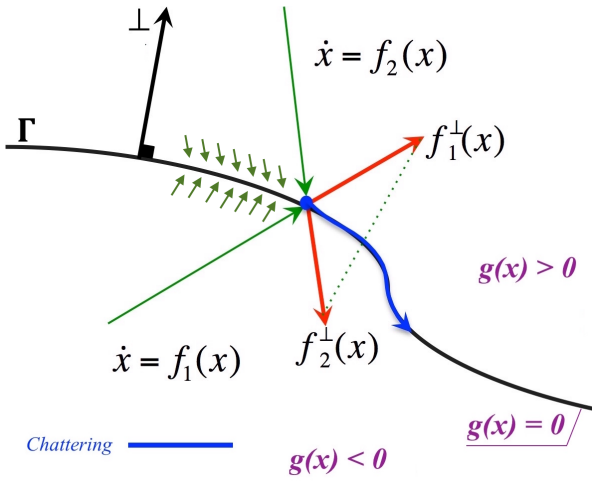

Figure 2. The chattering between two dynamics along a switching surface.

$\dot{x} = f_j(t,x)$ can be replaced by a differential inclusion $\dot{x} \in \eta(x)$ given as a convex set containing all the limit values of $f_j(x)$ for small neighbor $x(t,x) \notin \Gamma_j$ approaching Γ_j from the both sides (Biák et al., 2013). Equation 3 can be replaced then by:

$$\eta_j \in \frac{1-\delta_j(g_j(t,x))}{2}\cdot f_{j1}(t,x) + \frac{1+\delta_j(g_j(t,x))}{2}\cdot f_{j2}(t,x) \qquad (19)$$

where $\delta_j(\cdot)$ is a multi-valued sign function given by:

$$\delta_j(g_j(t,x)) = \left\{ \begin{array}{cc} -1 & for\ \ g_j(t,x) < 0 \\ (-1,\ 1) & for\ \ g_j(t,x) = 0 \\ 1 & for\ \ g_j(t,x) > 0 \end{array} \right\} \quad (20)$$

Roughly speaking, when a chattering occurs on Γ_j, we seek a smooth function δ_j, taking the value $\delta_j(g_j(t,x)) \in (-1,1)$, so that the new equivalent chattering-free dynamics $f_{j_{CHF}}$ is given for $\delta_j(g_j(t,x)) \in (-1,1)$ by:

$$f_{j_{CHF}} = \frac{1-\delta_j(g_j(t,x))}{2} \cdot f_{j1}(x) + \frac{1+\delta_j(g_j(t,x))}{2} \cdot f_{j2}(x) \quad (21)$$

The idea behind forcing the solution trajectory to stay on the swicthing manifold during chattering execution is by forcing the normal projection of the equivalent chattering-free dynamics onto the swicthing manifold Γ_j to be tangential to Γ_j, that is,

$$f^{\perp}_{j_{CHF}}(t,x) = \left(\frac{\partial g_j(t,x)}{\partial x} \right) \cdot f_{j_{CHF}}(t,x) = 0 \quad (22)$$

which implies

$$\left[\frac{1-\delta_j(g_j(t,x))}{2} \quad \frac{1+\delta_j(g_j(t,x))}{2} \right] \cdot \left[\begin{array}{c} f^{\perp}_{j1}(t,x) \\ f^{\perp}_{j2}(t,x) \end{array} \right] = 0 \quad (23)$$

and then

$$\delta_j(g_j(t,x)) = \frac{f^{\perp}_{j1}(t,x) + f^{\perp}_{j2}(t,x)}{f^{\perp}_{j1}(t,x) - f^{\perp}_{j2}(t,x)} \quad (24)$$

where $f^{\perp}_{j1}(t,x)$ and $f^{\perp}_{j2}(t,x)$ are given in equation 12 and equation 13, respectively.

By the substitution of equation 24 in equation 21, the equivalent chattering-free dynamics is given then by:

$$f_{j_{CHF}}(t,x) = \frac{f^{\perp}_{j1}(t,x) \cdot f_{j2}(t,x) - f^{\perp}_{j2}(t,x) \cdot f_{j1}(t,x)}{f^{\perp}_{j1}(t,x) - f^{\perp}_{j2}(t,x)} \quad (25)$$

A smooth exist from sliding takes place instantly at the time instant at which the sufficient condition of chattering is no longer satisfied, that is, when either $f^{\perp}_{j1}(t,x)$ or $f^{\perp}_{j2}(t,x)$ starts to change its signs (i.e. when either $f^{\perp}_{j1}(t,x) = 0$ or $f^{\perp}_{j2}(t,x) = 0$).

Once a chattering execution is detected during the simulation process, the following Algorithm 1 is employed to generate the chattering-free dynamics internally in the simulation loop of the simulator. The number of iterations need to be performed by Algorithm 1 to compute the chattering-free dynamics is equal to the total number of the switching manifolds Γ_j on which the chattering occurs instantly. That is, when the system chatters between two dynamics, i.e. a chattering onto a single switching manifold (as in Example 1), the equivalent chattering-free dynamics will be generated by Algorithm 1 in one iteration.

The main benefit of the iterative approach of Algorithm 1 is that it allows us to eliminate chattering efficiently in run-time simulation without any need to modes enumeration, even when the chattering is occurring on the intersection $\Delta = \bigcap_j(\Gamma_j), j = 1,2,...,p$ of a large number p of intersected switching manifolds. Another benefit is that there is no need to solve stiff nonlinear equations for the computation of the chattering-free coefficients $\delta_j(g_j(t,x))$ in case of chattering on switching intersection with $p > 1$.

Data: Discontinuous dynamics $f(t,x)$, swicthing functions $g_j(t,x)$.
Result: $f_{\Delta_{CHF}}(t,x) = f_{j_{CHF}}(t,x)$
Initialization:
$j = 1$;
$f(x(t)) = f_j(x(t))$ (equation 9);
while $j \leq p$ **do**
  Use $f_j(x(t))$ to build a differetial inclusion η_j (equation 19);
  Compute $f_{j_{CHF}}(t,x)$ (equations 21 to 25);
  Set $f_j(x(t)) = f_{j_{CHF}}(t,x)$;
  $j = j + 1$;
  Repeat;
end

Algorithm 1: How to generate the equivalent chattering-free dynamics $f_{\Delta_{CHF}}(t,x)$.

In the following two simple examples we illustrate the functionality of Algorithm 1 in case of chattering on switching intersection.

Example 2:

Consider the simplest case of chattering onto the intersection of two switching manifolds, Γ_1 and Γ_2, defined as the zeros of a set of scalar functions $g_1(t,x) = x_1(t)$ and $g_2(t,x) = x_2(t)$, respectively.

$\dot{x}_1 = 0$ **init** -$sgn(g_{10})$ **reset** [-1;1] **every up**[g_1;-g_1]
$\dot{x}_2 = 0$ **init** -$sgn(g_{20})$ **reset** [-1;1] **every up**[g_2;-g_2]
$g_1 = x_1$ **init** g_{10}; $g_2 = x_2$ **init** g_{20}

where the zero-crossing is described as an expression of the form **up**(z) that becomes true when the sign of the event function $z(t,x)$ switches from negative to positive during an execution, that is, **up**(z)= *True* if $z(t_{i-1},x_{i-1}) \leq 0 \wedge z(t_i,x_i) > 0$ (Schrammel, 2012). In this example, the trajectories initialized outside the origin reach the origin in finite time and with an infinite number of crossings of the switching surfaces $x_1(t) = 0$ and $x_2(t) = 0$. The finite time convergence is easy to establish as the time intervals between two switches satisfy a geometric series and consequently have a finite sum. This system has also an infinity of spontaneous switches from the origin, that is, there is an infinity of trajectories which start with the initial data (0,0), and except for the trivial solution that stays at the origin, they all cross the switching surfaces an infinity of times.

DOI 10.3384/ecp1612495

To generate the intersection chattering-free dynamics $f_{\Delta_{CHF}}(t,x)$ on the intersection (the origin) $\Delta = \Gamma_1 \cap \Gamma_2$, Algorithm 1 performs two iterations:

- In **Iteration1**, the algorithm computes the equivalent chattering-free dynamics on Γ_1 (equation 26).
- In **Iteration2**, the algorithm computes the equivalent chattering-free dynamics on the intersection $\Delta = \Gamma_1 \cap \Gamma_2$ (equation 27).

$$f_{1_{CHF}}(t,x) = \begin{bmatrix} 0 \\ \left\{ \begin{array}{rl} -1 & for\ x_2(t) > 0 \} \\ 1 & for\ x_2(t) < 0 \} \end{array} \right. \end{bmatrix} \quad (26)$$

$$f_{\Delta_{CHF}}(t,x) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (27)$$

Example 3: Stick-Slip Frictional System
Consider the following non-smooth mechanical system with friction elements.

$$f(x) = \left\{ \begin{array}{l} \dot{x}_{m_1} = v_{m_1} \\ \dot{v}_{m_1} = \frac{1}{m_1}\mathscr{F}_1 \\ \dot{x}_{m_2} = v_{m_2} \\ \dot{v}_{m_2} = \frac{1}{m_2}(u - kx_{m_2} - \mathscr{F}_1 - \mathscr{F}_2) \\ \dot{x}_{m_3} = v_{m_3} \\ \dot{v}_{m_3} = \frac{1}{m_3}\mathscr{F}_2 \end{array} \right\} \quad (28)$$

In this example, the entire disontinuity region is given as the union of two transversally intersected swicthing manifolds Γ_1 and Γ_2 defined as the zeros of a set of the scalar functions $g_1(t,x) = v_{m_2}(t) - v_{m_1}(t)$ and $g_2(t,x) = v_{m_2}(t) - v_{m_3}(t)$, respectively.

$\mathscr{F}_1 = 0$ **init** $F_{c_1} sgn(g_{10})$ **reset** $[F_{c_1}; \text{-}F_{c_1}]$ **every up**$[g_1; \text{-}g_1]$
$\mathscr{F}_2 = 0$ **init** $F_{c_2} sgn(g_{20})$ **reset** $[F_{c_2}; \text{-}F_{c_2}]$ **every up**$[g_2; \text{-}g_2]$
$g_1(t,x) = v_{m_2}(t) - v_{m_1}(t)$ **init** g_{10}
$g_2(t,x) = v_{m_2}(t) - v_{m_3}(t)$ **init** g_{20}

We have $p = 2$ intersected swicthing manifolds. The algorithm, then, performs two iterations to generate $f_{\Delta_{CHF}}(t,x)$.

The output of **Iteration1**:

$$f_{1_{CHF}}(t,x) = \left\{ \begin{array}{l} \dot{x}_{m_1} = v_{m_1} \\ \dot{v}_{m_1} = \frac{1}{m_1+m_2}(u - kx_{m_2} - \mathscr{F}_2) \\ \dot{x}_{m_2} = v_{m_2} \\ \dot{v}_{m_2} = \frac{1}{m_1+m_2}(u - kx_{m_2} - \mathscr{F}_2) \\ \dot{x}_{m_3} = v_{m_3} \\ \dot{v}_{m_3} = \frac{1}{m_3}\mathscr{F}_2 \end{array} \right\} \quad (29)$$

The output of **Iteration2**:

$$f_{\Delta_{CHF}}(t,x) = \left\{ \begin{array}{l} \dot{x}_{m_1} = v_{m_1} \\ \dot{v}_{m_1} = \frac{1}{m_1+m_2+m_3}(u - kx_{m_2}) \\ \dot{x}_{m_2} = v_{m_2} \\ \dot{v}_{m_2} = \frac{1}{m_1+m_2+m_3}(u - kx_{m_2}) \\ \dot{x}_{m_3} = v_{m_3} \\ \dot{v}_{m_3} = \frac{1}{m_1+m_2+m_3}(u - kx_{m_2}) \end{array} \right\} \quad (30)$$

4 Generic Implementation Scheme in FMI 2.0

In this section, a prototype implementation is sketched for applying the chattering-free computational framework from the previous section to Functional Mock-Up Interface v2.0 for Model Exachange. The goal is to provide in FMI, a rigorous chattering-free simulation, in run-time, without modes enumeration, for any chattering FMU which may be either generic or generated from a modeling environment in which chattering models can not be simulated rigorously, whenever the compliance with FMI specification for model exchange is fulfilled. The FMI chattering-free implementation has been performed by embeding the chattering detection and elimination algorithm in the Event Mode of the FMI.

4.1 The Functional Mock-Up Interface FMI

FMI is an open standard for model exchange and co-simulation between multiple software systems. This new standard, resulting from the ITEA2 project MODELISAR, in 2010, is a response to the industrial need to connect different environments for modeling, simulation and control system design. It is used to create an instance of a model which can be loaded into any simulator providing an import function for FMI. A software instance compatible to the FMI is called an FMU. An FMU is distributed as a compressed archive with a .fmu file extension. It contains a concrete mathematical model described by differential, algebraic and discrete equations with possible events of a dynamic physical system. An FMU consists basically of two parts:

- an XML format for model interface information,
- C API model interface functions according to the FMI specification, for model execution.

The XML format, specified by an XML schema conforming to the FMI specification, contains all static information about model variables, including names, units and types, as well as model meta data. The C API, on the other hand, contains C functions for data management, as setting and retrieving parameter values, and evaluation of the model equations. The implementation of the C API may be provided either in C source code format or in binary forms (e.g. in the form of Windows dynamic link library .dll or a Linux shared object library .so files) to protect the model developer's intellectual property. Additional parts can be added and compressed into the FMU, as the documentation and the icon of the model. FMUs can be written manually or can be generated automatically from a modelling environment.

4.2 Chattering-Free Support in FMI

In this section we explain the functionality of our chattering-free FMI framework as well as how the chattering behavior is treated internally in the main simulation loop of interface without any need to add hysterisis to the event indicators in the FMU.

Prior to a simulation experiment, the model has to be instantiated. This includes extracting the files in the FMU, loading the DLL and XML files and calling the instantiation function available in the DLL. A model can be instantiated multiple times for which the function `fmi2SetupExperiment` is provided.

Simulating an FMI model means to split the solution computation process in three different phases, categorized according to three modes: Initialization Mode, Continuous-Time Mode, and Event Mode.

In the Initialization Mode, the model is initialized with $f_{init}(\cdots)$ by calling the FMI function `fmi2EnterInitializationMode` in order to compute the continuous-time states and the output variables at the initial time t_0. There are FMI functions used in this Mode as `fmi2GetContinuousStates` as well as functions for setting and getting values for Type `Real`, `Integer`, `String`, and `Boolean` values, of the form `fmi2(Get/Set)`(Type). The input arguments to the Initialization Mode functions consist of the all variables that are declared with "input" and "independent" causality in the FMU XML files, as well as all variables that have a start value with initial = "exact". Once the model is instantiated and initialized it can be simulated.

The main simulation loop starts once the FMI function `fmi2ExitInitializationMode` is called. The simulation is performed by calculating the derivatives and updating time and states in the model via the FMI functions `fmi2SetContinuousStates`, `fmi2SetTime`, `fmi2GetContinuousStates`, `fmi2GetDerivatives`, as well as the four `fmi2(Get/Set)`(Type) functions mentioned above. To retrieve or set variable data during a simulation, value-references are used as keys. All variables are connected to a unique number defined and provided in the FMU XML-file. This number can then be used to retrieve information about variables via functions in the interface or can be used to set input values during a simulation. During the simulation, events are monitored via the functions `fmi2GetEventIndicators` and `fmi2CompletedIntegratorStep`. Events are always triggered from the environment in which the FMU is called, so they are not triggered inside the FMU (Blochwitz et al., 2012). Step-events are checked in the model after calling the completed step function `fmi2CompletedIntegratorStep` when an integration step was sucessfully completed. A step event occurs if indicated by the return argument nextMode = EventMode. For capturing state events during continuous integration, the algorithm monitors, at every completed integrator step, the set of event indicator functions $z_j(t,x)$ provided in the function `fmi2GetEventIndicators`. All event indicators $z_j(t,x)$ are piecewise continuous and are collected together in one vector of real numbers (Blochwitz et al., 2012). A state event occurs when the event indicator changes its domain from $z_j(t,x) > 0$ to $z_j(t,x) \leq 0$ or from $z_j(t,x) \geq 0$ to $z_j(t,x) < 0$. If a domain change of one of the indicator functions is detected, a state event has occurred and the simulation environment then informs the FMU by calling the function `fmi2NewDiscreteStates`.

During the continuous integration, we distinuiush, for each time integration step, the following cases:

1. If $z_j(t_i,x_i) \cdot z_j(t_{i+1},x_{i+1}) > 0$ for all $j = 1,2,...,p$ where p is the total number of the event indicators, then we continue integrating the system with the same dynamics.

2. If there exist $j \in \{1,2,...,p\}$ for which: $\forall \tau \in [t_i,t_{i+1}[: z_j(\tau,x) < 0 \ \wedge \ \exists\, m \leq margin : \ \forall \tau \in [t_{i+1},t_{i+1}+m] : z_j(\tau,x) \geq 0$, or $\forall \tau \in [t_i,t_{i+1}[: z_j(\tau,x) > 0 \ \wedge \ \exists\, m \leq margin : \forall \tau \in [t_{i+1},t_{i+1}+m] : z_j(\tau,x) \leq 0$, a zero crossing in the time interval $[t_i,t_i+1]$ is then detected. The algorithm performs an iteration over time between the previous and the actual completed integrator step, in order to determine the time instant of the switching point up to a certain precision. In this case we have a continuous smooth switching function $z_j(t_{i+1}(\sigma))$ taking opposed signs at $\sigma = 0$ and $\sigma = 1$ and therefore there exist a zero at $\sigma_e \in (0,1)$ which defines the state event $x_e = x_{i+1}(\sigma_e) \in \Gamma_j$, where $\Gamma_j = \{x \in \mathbb{R}^n | \ z_j(t,x) = 0\}$ is the switching surface.

3. The case in which there exist finitely many event indicator functions $z_j(t,x)$, $j \in \{1,2,...,p\}$, all satisfy: $\forall \tau \in [t_i,t_{i+1}[: z_j(\tau,x) < 0 \ \wedge \ \exists\, m \leq margin : \forall \tau \in [t_{i+1},t_{i+1}+m] : z_j(\tau,x) \geq 0$, or $\forall \tau \in [t_i,t_{i+1}[: z_j(\tau,x) > 0 \ \wedge \ \exists\, m \leq margin : \forall \tau \in [t_{i+1},t_{i+1}+m] : z_j(\tau,x) \leq 0$, and $z_j(\sigma_e) = 0$ for all $j = 1,2,...,k$ where $k \leq p$ and $\sigma_e \in (0,1)$, indicates that the solution trajectory has reached the intersection of $k \leq p$ of transvrsally intersected $\mathbb{R}^{(n-1)}$ switching manifolds Γ_j.

At an event, the function `fmi2NewDiscreteStates` has to be called. This function updates and re-initializes the model in order for the simulation to be continued. Information is also given about if the states have changed values, if new state variables have been selected and information about upcoming time events.

In our chattering-free semantics, the master algorithm has to decide, at the state event, whether the solution trajctory should cross the switching surface transversally or slide on it (to eliminate chattering). The computation of the chattering-free solution is split in two phases: i) chattering detection, and ii) chattering elimination.

DOI 10.3384/ecp1612495

The chattering detection phase starts once a state event is detected and located. The algorithm inspects whether the state event is a chattering event or not. This implies checking, at the state event, whether or not the sufficient condition of chattering is satisfied, by analyzing the gradients of the continuous time behavior before and after the state event. For doing so, the directional derivatives of the dynamics (the normal projection of dynamics onto the switching surface) should be computed and evaluated at the beginning and at the end of the completed integration step at which the state event has been detected. A state event $x_e \in \Gamma_j$ detected in the time interval $[t_i, t_{i+1}]$ is said to be a chattering event if the condition: $NP_j(t_i,x_i) \cdot NP_j(t_{i+1},x_{i+1}) < 0$ is satisfied, where $NP_j(t_i,x_i) = f_{j1}^{\perp_j}(t_i,x_i)$, respectively $NP_j(t_{i+1},x_{i+1}) = f_{j2}^{\perp_j}(t_{i+1},x_{i+1})$, is the normal projection of the dynamics f_{j1} (before the state event), respectively f_{j2} (after the state event), onto the switching manifold $\Gamma_j = \{x \in \mathbb{R}^n |\ z_j(t,x) = 0\}$, at t_i, respectively t_{i+1}. A chattering occurs on a switching intersection $\Delta = \bigcap_j \Gamma_j$ (i.e. intersection state event), detected in the time interval $[t_i, t_{i+1}]$, if for all $j = 1,2,...,k$: $NP_j(t_i,x_i) \cdot NP_j(t_{i+1},x_{i+1}) < 0$, where as mentioned in Section 4 (equation 12 and equation 13), the normal projection $NP_j(t_i,x_i)$, respectively $NP_j(t_{i+1},x_{i+1})$, is computed as a scalar product of the dynamics $f(t_i,x_i)$, respectively $f(t_{i+1},x_{i+1})$, with the partial derivatives the event indicator function z_j. The partial derivatives of z_j are computed numerically in the integration step $[t_i, t_{i+1}]$ at which the state event is detected. As the nature of our chattering detection semantics is to compare the sign of the directional derivatives (normal projections) at the beginning and the end of the time integration step $[t_i, t_{i+1}]$ in which a state event is occurred, and if it changes, declare a chattering event, the environment then should be able to have an access to the dynamics at t_i (i.e. in the previous domain before the event) and at t_{i+1} (i.e. in the next domain after the event). For doing so, we use two arrays, $xdot_{pre}$ and $xdot_{post}$, where during the continuous integration, and for each time step $[t_i, t_{i+1}]$ in which a state event has been detected, the dynamics $f(t_i,x_i)$, and $f(t_{i+1},x_{i+1})$ are computed and evaluated via `fmi2GetDerivatives` and then stored in $xdot_{pre}$, and $xdot_{post}$, respectively. In the chattering elimination phase, Algorithm 1 (Section 3) is employed in the environment's master algorithm in order to compute the smooth equivalent chattering-free dynamics internally giving the dynamics before and after the state event, f_{j1} and f_{j2}, repectively, as well as the event indicators $z_j(t,x)$. Once the solution is at the final time of a simulation, the function `fmi2Terminate` is called to terminate the simulation. After a simulation is terminated, memory has to be deallocated. The function `fmi2FreeInstance` is then called to deallocate all memory that have been allocated since the initialization.

5 Simulation Results

Figure 3 shows the chattering-free simulation of the system in Example 1 for the data set: $\beta = 0.5$, $x_0 = [0.5, 3, 1]^T$. During a simulation time $t = 10$, 241685 chattering events have been detected and replaced by two sliding windows. The first chattering event is detected at $t = 2.649$ (Figure 4), the algorithm switches to integrate the the system with the chattering-free dynamics generated internally. In Figure 5 and Figure 6, the Stick-Slip frictional systm in Example 3 was simulated for $m_1 = m_2 = m_3 = 1[kg]$, $k = 0.88[N \cdot m^{-1}]$, $F_{c_1} = 0.01996[N]$ $F_{c_2} = 0.062[N]$, and $x_0 = [0.8295\ 0.8491\ 0.3725\ 0.5932\ 0.8726\ 0.9335]^T$. The external force u was simulated as a sine wave of frequency of $\omega = 0.073[rad/sec]$. The sliding bifurcations depend on the effect of the external force u and the level of Coulomb frictions F_{c_1} and F_{c_2}. At the time instant $t = 32.69$ *sec*, two masses m_2 and m_3 stick together and the solution trajectory start a sliding motion on the switching manifold $\Gamma_2 = \{x \in \mathbb{R}^n :\ (v_{m_2}(t) - v_{m_3}(t) = 0)\}$ (Figure 5). A smooth exit from sliding on Γ_2 to evolve into q_3 was detected at the time instant $t = 77.23$ *sec*. A transversality switching from the discrete state q_3 to the discrete state $q_1 = \{x \in \mathbb{R}^n :\ (v_{m_2}(t) - v_{m_1}(t) > 0) \wedge (v_{m_2}(t) - v_{m_3}(t) > 0)\}$ at the intersection $\Delta = \Gamma_1 \cap \Gamma_2$ was detected at $t = 92.04$ *sec*.

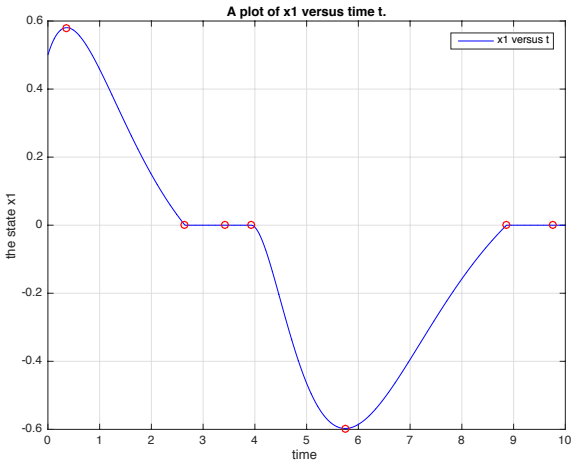

Figure 3. The time evolution of the continuous state x_1 with chattering-free simulation.

6 Conclusions

In this paper we presented an FMI-based computational framework, and a prototypical implementation of a generic chattering-free FMI for robust and reliable detection and elimination "On the Fly" of chattering behavior in run-time simulation of non-smooth hybrid systems, without modes enumeration, and without any need to add a small hysteresis to the event indicators in the FMUs.

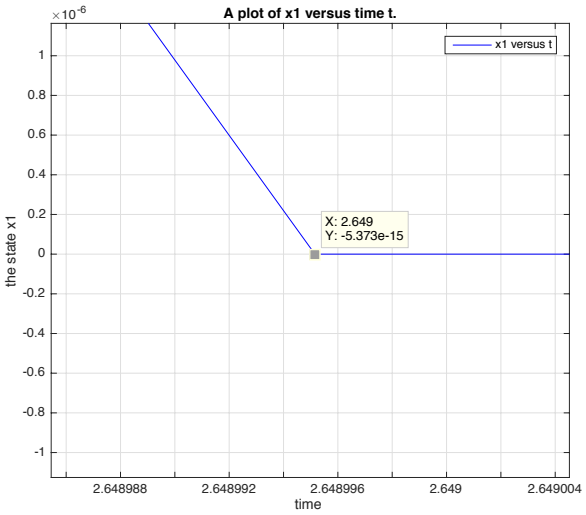

Figure 4. A smooth entering to sliding: First chattering state event detected at $t = 2.649$.

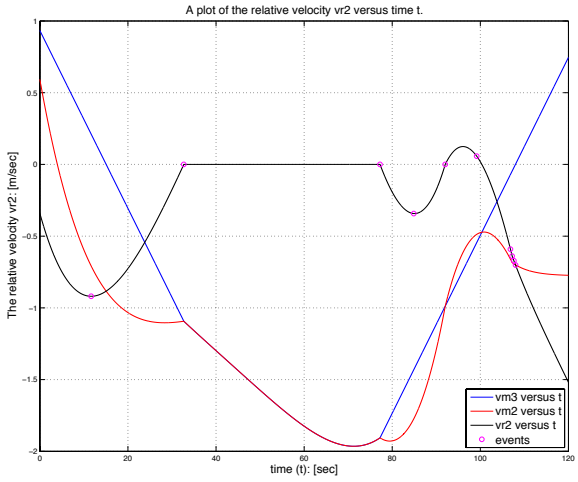

Figure 5. A chattering-free simulation of Example 3: The time evolution of the relative velocity $v_{m_2}(t) - v_{m_3}(t)$.

The developed chattering-free FMI switches between the transversality modes and the sliding modes simulation automatically, integrates each particular state appropriately, and localizes the non-smooth structural changes in the system in an accurate way. It treats the chattering non-smoothness in the trajectory of the state variables by a smooth correction after each integration time-step. Our chattering-free FMI can robustly handle the case of chattering on switching intercsetion without any need to solve stiff nonlinear equations for the computation of the chattering-free coefficients. Furthermore, a guidance for development of a hybrid chattering-free version of the FMI standard, was provided in this paper. Finally, the simulation results on a set of representative examples have demonstrated that our FMI-based chattering-free framework is efficient and precise enough to provide a

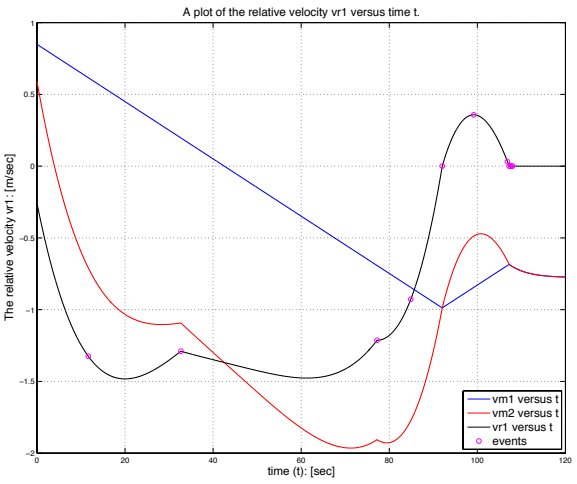

Figure 6. A chattering-free simulation of Example 3: The time evolution of the relative velocity $v_{m_2}(t) - v_{m_1}(t)$.

rigorous chattering-free simulation for any generic chattering Functional Mockup Unit (FMU) conforming to the FMI standard v2.0 Specification for model exchange.

Acknowledgements

This work was supported by the ITEA2 MODRIO project under contract N^{o} 6892, and the ARED grant of the Conseil Régional de Bretagne.

References

Ayman Aljarbouh and Benoit Caillaud. On the regularization of chattering executions in real time simulation of hybrid systems. *Baltic Young Scientists Conference Proceedings*, pages 49–66, 2015a. URL `https://hal.archives-ouvertes.fr/hal-01246853v2`.

Ayman Aljarbouh and Benoit Caillaud. Robust simulation for hybrid systems: Chattering path avoidance. *Linköping Electronic Conference Proceedings*, 119(018):175–185, 2015b. ISSN 1650-3686. doi:10.3384/ecp15119175. URL `http://www.ep.liu.se/ecp/119/018/ecp15119018.pdf`.

Martin Biák, Tomáš Hanus, and Drahoslava Janovská. Some applications of filippov's dynamical systems. *Journal of Computational and Applied Mathematics*, 254:132–143, 2013. ISSN 0377-0427. doi:http://dx.doi.org/10.1016/j.cam.2013.03.034. URL `http://www.sciencedirect.com/science/article/pii/S0377042713001428`.

Torsten Blochwitz, Martin Otter, Johan Akesson, Martin Arnold, Christoph ClauB, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. *In Pro-*

ceedings of 9th International Modelica Conference, Munich, Germany, 076(017):173–184, 2012. ISSN 1650-3686. doi:10.3384/ecp12076173. URL http://www.ep.liu.se/ecp/076/017/ecp12076017.pdf.

Chaohong Cai, Rafal Goebel, Ricardo Sanfelice, and Andrew Teel. *Hybrid systems: limit sets and zero dynamics with a view toward output regulation*. Springer-Verlag, 2008. URL https://hybrid.soe.ucsc.edu/files/preprints/21.pdf.

Mario di Bernardo, Chris J. Budd, Alan R. Champneys, Piotr Kowalczyk, Arne B. Nordmark, Gerard Olivar Tost, and Petri T. Piiroinen. Bifurcations in nonsmooth dynamical systems. *SIAM Review*, 50(4):629–701, 2008. doi:10.1137/050625060. URL http://dx.doi.org/10.1137/050625060.

A.F. Filippov. *Differential Equations with Discontinuous Righthand Sides*. Springer Netherlands, 1988. ISBN 978-94-015-7793-9.

K.H. Johansson, A.E. Barabanov, and K.J. Astrom. Limit cycles with chattering in relay feedback systems. *Automatic Control, IEEE Transactions on*, 9(018):1414–1423, 2002. ISSN 0018-9286. doi:10.1109/TAC.2002.802770. URL http://www.ep.liu.se/ecp/119/018/ecp15119018.pdf.

Remco I. Leine and Henk Nijmeijer. *Dynamics and Bifurcations of Non-Smooth Mechanical Systems*. Springer Berlin Heidelberg, 2004. ISBN 978-3-642-06029-8.

John Lygeros, Claire Tomlin, and Shankar Sastry. *Hybrid Systems: Modeling, Analysis and Control*. Lecture Notes on Hybrid Systems, 2008. URL http://inst.cs.berkeley.edu/~ee291e/sp09/handouts/book.pdf.

Peter Schrammel. *Logico-Numerical Verification Methods for Discrete and Hybrid Systems*. PhD dissertation, 2012.

Vadim I. Utkin. *Sliding Mode in Control and Optimization*. Springer Berlin Heidelberg, 1992. ISBN 978-3-642-84379-2.

D. Weiss, T. Küpper, and H.A. Hosham. Invariant manifolds for nonsmooth systems with sliding mode. *Mathematics and Computers in Simulation*, 110:15–32, 2015. doi:http://dx.doi.org/10.1016/j.matcom.2014.02.004. URL http://www.sciencedirect.com/science/article/pii/S037847541400041X.

Jun Zhang, Karl Henrik Johansson, John Lygeros, and Shankar Sastry. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control*, 11(5):435–451, 2001. ISSN 1099-1239. doi:10.1002/rnc.592. URL http://dx.doi.org/10.1002/rnc.592.