Simulation Modelling Practice and Theory 68 (2016) 1–17

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A zero-crossing detection algorithm for robust simulation of hybrid systems jumping on surfaces

David A. Copp[a,*], Ricardo G. Sanfelice[b,*]

[a] Department of Mechanical Engineering, University of California, Santa Barbara, CA 93106-9560, USA
[b] Department of Computer Engineering, University of California, Santa Cruz, CA 95064, USA

ARTICLE INFO

Article history:
Received 4 March 2016
Revised 27 May 2016
Accepted 18 July 2016

Keywords:
Hybrid systems
Zero-crossing detection
Simulation

ABSTRACT

Hybrid systems are inherently fragile with respect to perturbations when their state experiences jumps on surfaces. Zero-crossing detection algorithms are capable of robustly detecting the crossing of such surfaces, but, up to now, the effects of adding such algorithms to the system being simulated are unknown. In this paper, we propose a mathematical model for hybrid systems that incorporates zero-crossing detection as well as a hybrid simulator for it. First, we discuss adverse effects that measurement noise and discretization can have on hybrid systems jumping on surfaces and prove that, under mild regularity conditions, zero-crossing detection algorithms can robustify the original system. Then, we show that integration schemes with zero-crossing detection actually compute a robustified version of the fragile nominal model. In this way, we rigorously characterize their effect on solutions to the simulated system. Finally, we show that both the model and simulator are not only robust, but also that the hybrid simulator preserves asymptotic stability properties, semiglobally and practically (on the step size), of the original system. Several examples throughout the paper illustrate these ideas and results.

1. Introduction

This work considers dynamical systems with a state that experiences instantaneous resets (jumps) when it hits a *switching surface* $\mathcal{S}$. A switching surface is typically defined as the zero-level set of a continuously differentiable function, defining in this way a codimension one submanifold of $\mathbb{R}^n$; see, e.g., [1–3]. The state is denoted by x and takes values from a region of operation $\mathcal{X} \subset \mathbb{R}^n$. When x is away from the surface $\mathcal{S}$, the continuous dynamics (flows) of the system are given by a differential equation. This can be written more precisely as

$$\dot{x} = f(x) \quad \text{when } x \in \mathcal{X} \setminus \mathcal{S}. \tag{1}$$

When x hits the surface $\mathcal{S}$ while in the region of operation $\mathcal{X}$, the state is reset via a difference equation which defines the jumps of the system. More precisely, the new value of x after the jump, denoted x^+, is determined by

$$x^+ = g(x) \quad \text{when } x \in \mathcal{S} \cap \mathcal{X}. \tag{2}$$

* Corresponding authors.
E-mail addresses: dacopp@engr.ucsb.edu (D.A. Copp), ricardo@ucsc.edu (R.G. Sanfelice).
URL: https://hybrid.soe.ucsc.edu (R.G. Sanfelice)

http://dx.doi.org/10.1016/j.simpat.2016.07.005
1569-190X/

In this way, the trajectories are allowed to flow when $x \in \mathcal{X} \setminus \mathcal{S}$ and are allowed to jump when $x \in \mathcal{S} \cap \mathcal{X}$. This model captures the dynamics of control systems in which a controller makes decisions when certain variables hit a surface. For instance, in reset control systems (see, e.g., [4–7]), the output of the controller is reset to zero whenever its input and output satisfy an algebraic condition. Reset controllers have been found useful in applications as they improve the performance of linear systems [4]. In state-dependent impulsive control systems, (see, e.g., [1], [8]), jumps occur when the state of the system belongs to a surface in the state space of the system. Impulsive controllers are widely used in robotics to switch among several feedback laws when the state of the system reaches a surface [9,10].

Several difficulties arise when dealing with systems jumping on surfaces. One such difficulty is with "grazing" of the flowing solution at the boundary of the switching surface $\mathcal{S}$ without crossing over it. This can lead to non-unique solutions and is known as the grazing phenomenon. For a discussion of methods to handle this phenomenon, see [11]. Further difficulties may be caused by perturbations due to measurement noise or numerical errors due to discretization. For example, suppose that the value of the state x of the system is perturbed when nearby $\mathcal{S}$ (e.g., due to measurement noise). Letting e denote this perturbation on x, the perturbed state can be written as $x + e$. Suppose that, for a given solution $x(t)$ to system (1) and (2) (using an appropriate notion of solution), e is zero when $x(t) \neq \mathcal{S}$ but equal to a nonzero constant ε when $x(t) = \mathcal{S}$. Then, when the perturbation e is added, for any nonzero ε, the same solution $x(t)$ will not satisfy the condition $x(t) + e(t) \in \mathcal{S}$, and therefore, will not jump at the instant that it would without noise. This suggests that arbitrarily small perturbations to (1) and (2) can generate trajectories that are nowhere close to the trajectories of the nominal system; see [12] for related discussions. Similarly, issues can arise from numerical errors introduced by discretization, such as numerical integration errors. In the presence of small numerical integration errors, which can be made arbitrarily small by adjusting the step size used in the simulator, trajectories obtained through simulation may never hit the surface, and therefore, never jump.

One way to resolve the issue of the numerical solution never hitting the switching surface, which is widely used in simulation packages, is to include *zero-crossing detection* (ZCD) algorithms to detect crossing of the surface $\mathcal{S}$. For instance, in MATLAB/Simulink, a block called *Hit Crossing* detects when the input of the block u reaches a specified offset parameter value u^* (other blocks and local ZCD options are also available in MATLAB/Simulink). The output of the block equals 1 when the value of the input has hit or crossed the offset parameter, or 0 otherwise. In fact, MATLAB/Simulink help files note the following [13]:

> "if the input signal is exactly the value of the offset value after the hit crossing is detected, the block continues to output a value of 1. If the input signals at two adjacent points bracket the offset value (but neither value is exactly equal to the offset), the block outputs a value of 1 at the second time step."

More precisely, denoting the step size by s, this block determines if the sign of $u - u^*$ at the integration time t has changed with respect to its value at the previous integration time $t - s$, i.e., it determines whether

$$(u(t) - u^*)(u(t - s) - u^*) \leq 0. \tag{3}$$

In order to determine whether this condition holds, a memory state must be added to keep track of the sign (relative to u^*) of the previous input to the block. While zero-crossing detection might be a remedy to detect the crossings of $\mathcal{S}$, a simulator utilizing ZCD actually modifies the original system by incorporating an extra mechanism for the detection of zero crossings. On the other hand, the relationship between the simulations obtained with ZCD and the true solutions of a system is not well understood.

The purpose of this paper is to introduce a mathematical framework for theoretical study of such algorithms and their effect in simulation of hybrid systems. To that end, the effect of perturbations to hybrid systems jumping on surfaces is highlighted. In particular, we point out that measurement noise and discretization due to numerical integration can lead to simulations that do not hit the surface and never jump. Following the ideas discussed above regarding zero-crossing detection algorithms used in software packages, we propose a mathematical model of a hybrid system incorporating a zero-crossing algorithm. We prove that, under mild regularity conditions, such a resulting hybrid system not only includes all of the nominal solutions (no perturbations) to the original system but is also robust to measurement noise. We argue that, rather than computing the solutions to the discretization of the fragile nominal model (1) and (2), integration schemes with zero-crossing detection actually compute the solutions of a robustified version of the fragile nominal model. Finally, we propose a hybrid simulator for the hybrid system with incorporated zero-crossing detection. As a difference to [14–16], we focus on detection of zero-crossing rather than accurate location, which is another important issue in simulating systems like (1) and (2). Even though we do not discuss finding accurate locations of zero-crossings, we show that our proposed hybrid simulator enjoys the following properties, which are illustrated in examples throughout the paper:

(1) Trajectories obtained with the proposed simulator with zero-crossing detection approximate the solutions to the original hybrid system with arbitrary precision. See Theorem 5.8.
(2) The proposed simulator with zero-crossing detection preserves the asymptotic stability properties of the original hybrid system. Furthermore, the proposed simulator with zero-crossing detection has an asymptotically stable compact set that converges to the asymptotically stable compact set of the original hybrid system. See Theorems 5.10 and 5.11.

An important feature of the proposed hybrid simulator is that it confers the above properties to the simulation of the original hybrid system by minimally affecting the original system. In fact, the only addition to the original system consists