Zero-Crossing Location and Detection Algorithms For Hybrid System Simulation

Fu Zhang * **Murali Yeddanapudi** * **Pieter J. Mosterman** *

* *The MathWorks, Inc., Natick, MA 01760, USA*

Abstract: Computational models of embedded control systems often combine continuous-time with discrete-event behavior, mathematically representing *hybrid dynamic systems*. An essential element of numerical simulation of a hybrid dynamic system is the generation of discrete events from continuous variables that exceed thresholds. In particular, the occurrence of such an event has to be *detected* and the point in time where the threshold is first exceeded has to be *located*. This paper presents a number of problems that are encountered in event detection and location when using existing techniques. Solution strategies that balance efficiency and robustness are presented to address: (i) repeated detection of a zero-crossing event at consecutive time steps, (ii) masked zero-crossing events because of multiple zero-crossing functions, and (iii) chattering and Zeno behavior.

Keywords: Zero crossings, Hybrid system, Zeno system, Simulation

1. INTRODUCTION

Modern engineered systems have reached a level of complexity that necessitates the use of computational models for their design. Models of system dynamics come in a variety of forms. They reflect physical behavior (e.g., power consumption and performance of hydraulic actuators) as well as controller behavior (e.g., controller output of discretized and scheduled computations realized in fixed point computation). So, the comprehensive behavior of an embedded control system inherently consists of two parts:

- A *controller* operates in a time discretized manner, often abstracted into a discrete-event representation.
- A *plant* operates based on principles of physics such as *conservation of energy* and *continuity of power* and is often best represented by continuous-time models.

The mathematical representation of models with continuous-time behavior and discrete-event behavior is referred to as a *hybrid system* or *hybrid dynamic system* (e.g., Vaandrager and van Schuppen [1999], Lynch and Krogh [2000], Benedetto and Sangiovanni-Vincentelli [2001]).

For dynamic systems, many analyses rely on *numerical simulation*. Simulation of continuous time, differential equation based models is well understood. The same holds for the simulation of discrete event models. The combination of these in hybrid dynamic systems, however, results in a number of idiosynchratic issues in the interaction that require dedicated facilities in the simulation engine (Mosterman [1999]).

In the most widely used paradigm (e.g., Guckenheimer and Johnson [1995]), continuous behavior operates in a mode, χ, and is governed by a set of differential equations, f_χ, or $\dot{x} = f_\chi(x, u, t)$, where the dot operator represents the derivative with respect to time; x is the vector of state variables; u is the vector of exogeneous variables; and t is the independent variable, time. The discrete event behavior can be represented by a finite state machine, ϕ, a four tuple $\phi =< \alpha, \delta, \chi_0, \chi >$, in which events δ, cause changes of the state, χ, and generate actions, α, while the initial state is given by χ_0. Interaction then takes place by generating events δ from the continuous variables by a relation g (i.e., $g(x, u, t) \rightarrow \delta$) and by changes in the differential equation, f_χ, because of discrete event state changes, χ. In some hybrid dynamic systems paradigms, the discrete event state change may also cause discontinuous changes in the continuous state variables, i.e., $x^+ = h_\chi(x, u, t)$, where x^+ is the new value and h_χ the function that specifies the change when the discrete state χ is reached.

Efficient simulation has to be explicitly concerned with the nature of the separate parts that comprise hybrid dynamic system behavior (Mosterman and Biswas [2002]):

- The continuous behavior is best described by differential equations, often in an explicit ordinary differential equation (ODE) form. Numerical integration routines solve these ODEs to generate trajectories of continuous behavior.
- The discrete event behavior is most efficiently handled by event-based simulators.
- Upon startup, a set of values has to be available to initialize the system state. Furthermore, whenever events occur, the state vector of the continuous-time model may change (in value and in dimension) and new initial values may have to be computed.
- Discrete events have to be generated from variables that represent behavior on a dense domain, typically based on threshold crossings. This requires a robust approach to detect if and when a threshold crossing occurs.

This paper concentrates on the part that generates discrete events from continuous variables. The presented technology will enable efficient modeling and accurate simulation of large models of physical systems that exhibit hybrid be-

havior such as the electro-hydraulic components in aircraft and power electronics systems. These systems tend to have a large number of modes (typically in the thousands) that are modeled using continuous signals that generate events.

Generating these events is generally implemented using relational operations (e.g., 'larger than'). For accurate simulation, the point in time at which these relations change their truth value has to be located within a small tolerance. To this end, a zero-crossing function of the general form $g(x, u, t) = 0$ can be used to identify the boundary at which the change takes place. Existing techniques for zero-crossing detection and location are inefficient and inaccurate when faced with handling such a large number of zero-crossing functions. For example, on many occasions the location mechanism is triggered twice: first when the zero-crossing function becomes 0 and second when the zero-crossing becomes nonzero. Furthermore, presently employed methods to handle zero crossings are susceptible to classes of pathological behaviors. This paper presents an approach to address these inefficiencies and to eliminate two classes of pathological behaviors.

Section 2 introduces the basic technology of zero-crossing detection and location. In Section 3 detection inefficiencies and methods to address these are presented. In Section 4 classes of behavior that are difficult to handle because of accurate event location are introduced and a solution is given. In Section 5 a benchmark case study illustrates the results obtained with the introduced solution. Section 6 provides conclusions of this work.

2. FUNDAMENTALS

The use of thresholding to model the interaction between the continuous time and discrete event model parts is intuitive and studied extensively (Cellier [1979], Shampine et al. [1991], Park and Barton [1996]). In this setup, continuous behavior is generated based on a differential equation $\dot{x} = f_\chi(x, u, t)$ while a subset of the state, input, and independent variables is used as argument to an indicator function $g_\chi(x, u, t)$. When this indicator function changes sign (i.e., its result changes from positive to negative or *vice versa*), a discrete event, δ, is generated. The event of the function changing its sign is referred to as a *zero-crossing*. A simple example of this setup can be written as:

$$\dot{x} = \begin{cases} f_1(x), g(x,t,u) \geq 0 \\ f_2(x), g(x,t,u) < 0 \end{cases} \quad (1)$$

where $g(x, t, u)$ is called a guard function, indicator function, or zero-crossing function.

Cellier [1979] introduced the *discontinuity locking* method to properly simulate systems with abrupt changes in continuous-time behavior. In general, this method contains the following stages, as depicted in Fig. 1:

(1) *Discontinuity Locking* Integration algorithms are typically based on the assumption that the states and their derivatives are continuous. As such, when a discontinuity occurs, special provisions must be taken to ensure the assumption holds true. This is implemented by not revealing the discontinuity to the integration algorithm until the discontinuity is located. The simulation step from T_{n-1} to t_n is a trial step. At t_n, $\dot{x}$ should switch to the value of f_2. However, to ensure the above assumption holds, f_1 is used till the integration step is accepted.

(2) *Zero-Crossing Detection* Checking the sign of the zero crossing function g at time T_{n-1} and t_n. If $g_{n-1} \times g_n \leq 0$, an event δ is detected unless $g_{n-1} = g_n = 0$.

(3) *Zero-Crossing Location* Once a zero crossing has been detected, it must be located within a tolerance. A direct approach computes the sensitivity of the zero crossing function with respect to the integrator step size (Esposito et al. [2001]). For linear systems, this allows direct computation of the point at which the zero crossing occurs.

In many cases, the functions required for this sensitivity computation are not explicitly available. The most robust indirect method in this case is a bisectional search as it finds the correct zero crossing if two functions g_1 and g_2 each have a zero crossing in the same interval. More efficient approaches (e.g., Moler [1997]) use sophisticated iteration schemes such as Newton-Raphson and to balance robustness and efficiency, different approaches may be combined.

During the search process, the zero crossing function g should be computed and thus the values of state x are needed. Usually these values are computed by interpolation, using the value X_{n-1} and x_n. Because of the nature of finite precision arithmetic on digital computers, the time that the event occurred can only be located within an interval $[T_L, T_R]$ that corresponds to machine precision. During each iteration, the zero-crossing function is evaluated twice: at the left and the right side of the reducing interval.

(4) *Event Handling* After the event is bracketed by T_L and T_R, the ODE solver first advances integration time from the previous time step T_{n-1} to T_L. The solver is then reset before advancing to T_R followed by switching the mode (in Fig. 1, switching the $\dot{x}$ to f_2). In doing so, the assumption of continuity holds throughout the numerical integration.

When faced with a large number of zero crossings, several issues associated with the above stages may arise. These issues and solution strategies are discussed next.

3. ISSUES WITH ZERO-CROSSING DETECTION

An important set of problems pertains to properly detecting whether a dense variable has exceeded a threshold and which of a potential set of variables achieves this first in a strict time ordering.

3.1 Even Roots

The nature of zero-crossing detection is to compare the sign of a function value, and if it changes, declare that it crossed zero. This approach may fail if the zero-crossing function has even zeros in between the two evaluated points, as determined by the numerical integration algorithm (see Fig. 2). In general, the zero crossing function g, is a function of the model state, but it does not contribute to its continuous dynamics, f. Therefore, numerical integration can proceed without taking the dynamics of g into

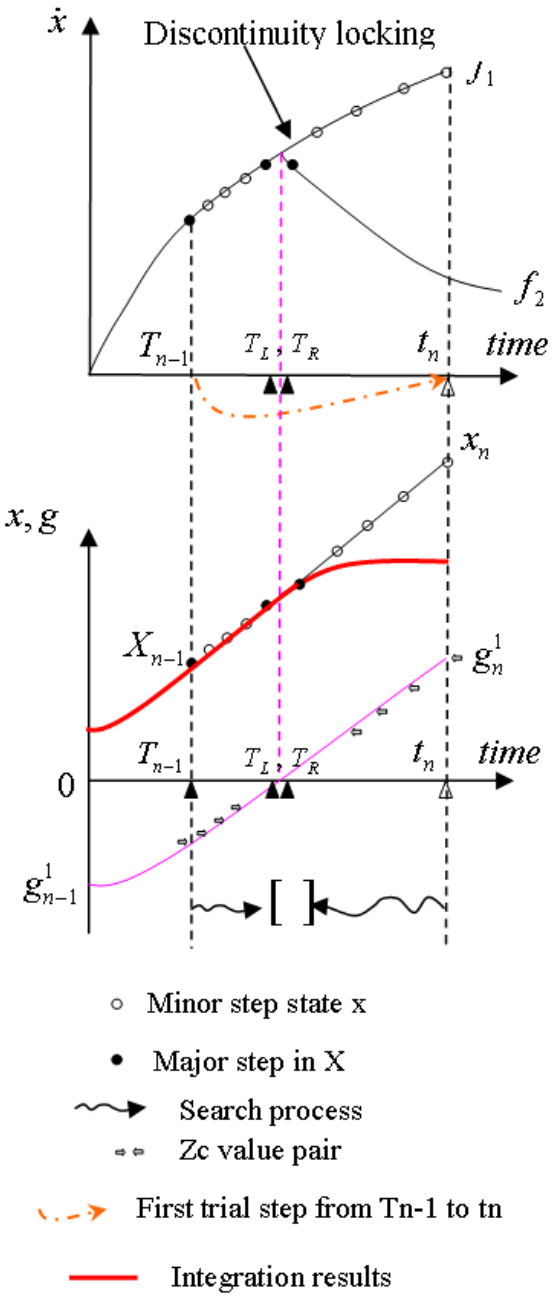


Fig. 1. Zero crossing detection and location stages

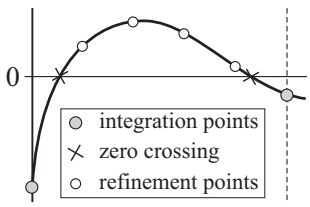


Fig. 2. Even roots problem

account, and when these are faster than the dynamics of f, the even roots situation may arise.

One solution to this is to include the g dynamics in the model dynamics so the numerical solver adjusts its step-size when too large an error (this will be caused by even zeros) is found (Park and Barton [1996]). Alternatively,

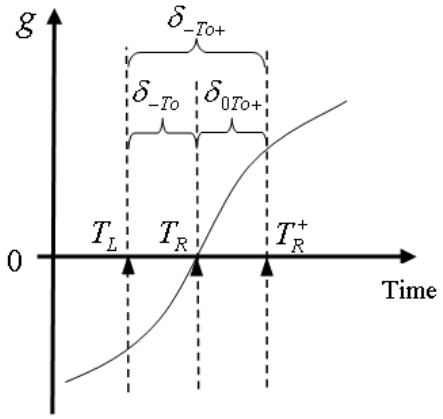


Fig. 3. Remove double zero crossing events

the sensitivity of the zero crossing function against the discrete step size can be computed and used to drive the step-size selection (Esposito et al. [2001]). In both cases, additional computations are required during the numerical integration.

Another method is to divide the intervals T_{n-1} and t_n into several smaller intervals and evaluate the zero crossings at the end of each interval. This method is called *zero crossing refinement* and reduces the likelihood of the even roots problem. Although this method does not guarantee eliminating the problem, it is in general computationally more efficient.

3.2 Double Detection

Another issue with zero crossing detection arises when the zero crossing function g returns exactly 0 at the right side T_R of the bracket. Once this happens, the solver first detects a '$-To0$' event [1] within $[T_L, T_R]$. When the simulation moves forwards from T_R, a '$0To+$' event between T_R and T_R^+ may be detected. Thus, two events (δ_{-To0} and δ_{0To+}) are reported consecutively, instead of one '$-To+$' event. This is a problem if the detected event triggers computation, because such computation will be executed twice where it should only be executed once.

To remove ouble event detection, the event δ_{-To+} can be defined as:

$$\delta_{UP} = \delta_{-To+} = \delta_{-To+} \mid \delta_{-To0} \mid \delta_{0To+} \qquad (2)$$

where | is a logical disjunction (the *OR* operator). Suppose an δ_{-To+} event (for example, a rising reset) is to be detected and if δ_{-To0} and δ_{0To+} are detected consecutively, these two events will be combined into one '$-To+$' event, as shown in Fig. 3.

3.3 Masked Even Roots

When there is more than one zero-crossing function in the system, the even roots problem may cause a side effect that is referred to as a *masked even roots* zero crossing. As

[1] The notation to indicate the type of zero-crossing event first states the original sign of the indicator function ('-', '0', or '+') then includes the string 'To' and finally the new sign of the indicator function.

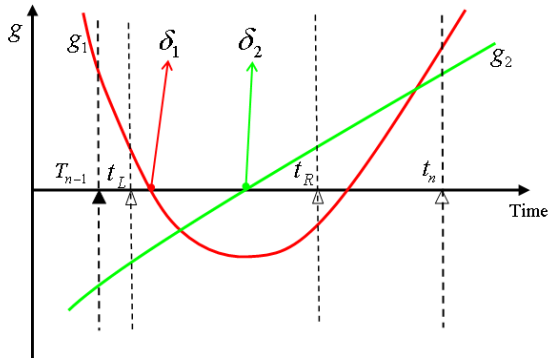


Fig. 4. Masked even roots problem

depicted in Fig. 4, there are two zero-crossing functions, g_1 and g_2, in the system. The solver moves from T_{n-1} to t_n. During this step, because g_1 has two roots in the interval $[T_{n-1}, t_n]$, the solver will detect that only g_2 had an event. Next, during the event location phase, the zero crossing function value of both g_1 and g_2 may be evaluated at t_L and t_R in one iteration. Between t_L and t_R there is no even roots problem, and δ_1 and δ_2 will be located. This seems to be good because both δ_1 and δ_2 have been found. However, suppose the system only has g_1, it is highly possible that δ_1 will not be found.

This will cause the following practical problem. Suppose that two subsystems containing g_1 and g_2, respectively, were simulated separately, and their results were recorded, studied and used as baselines. If these two subsystems are integrated into one system, the output of each subsystem may be different from the results in isolation, because of the masked zero crossing problem. Therefore, if a masked zero crossing is detected, the user should be informed. The option to avoid this is to increase the zero crossing refinements to reduce the chance of even roots to happen.

4. ISSUES WITH ZERO-CROSSING LOCATION

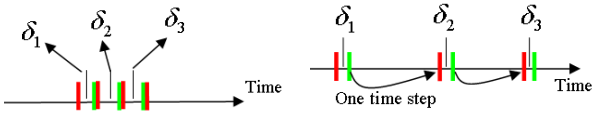
Three classes of pathological behaviors of hybrid dynamic systems can be identified: (i) a loop of discrete state changes may emerge, (ii) repeated switching between two modes with infinitesimal time spent between may occur, and (iii) discrete events may occur increasingly close in time, converging against a limit point. In all of these cases, numerical simulation effectively stops progressing in logical time. A solution strategy to the latter two classes is proposed.

The case of repeated switching between modes arises when the gradient of continuous-time behavior in each of the modes is directed towards the switching surface between the two modes. When in either of the modes on the 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. This behavior is sometimes referred to as *chattering*.

While the infinitely fast switching between modes occurs, a relatively slow motion along the switching surface may emerge. Dedicated simulation algorithms have been developed to derive the slow behavior by simulation while eliminating the fast chattering (e.g., Mosterman et al. [1999]). These algorithms have limited applicability, however.

The case of a series of events that converge against a limit point is sometimes referred to as *Zeno* behavior (e.g., Johansson et al. [1999], Zheng [2006]). In models that exhibit such behavior, an event occurs at an increasingly smaller distance in time. For example, if a new event occurs after half the time between the two previous events, a series of events emerges that, after n events, has moved in time according to $\sum_{k=1}^{n} \frac{1}{2^k}$. This series converges against 1 in the limit of $n \to \infty$.

Although analytically distinctly different, the effect of chattering and Zeno behavior during numerical simulation is similar; the simulation appears to come to a halt. This is illustrated in Fig. 5 for the two cases. Figure 5(a) shows how chattering causes the previous T_R to become the T_L of the next time step, and the integration will move with the minimum step size allowed. Figure 5(b) shows how Zeno behavior causes a mode switch to occur at each simulation time step. In other words, each major integration step is at one end of an interval that locates an event.



(a) Chattering (b) Zeno

Fig. 5. Classes of pathological behavior

A solution strategy to regularize these two classes of pathological behaviors is proposed. The results are illustrated using a variant of an extensively studied system with Zeno behavior, the bouncing ball model (Johansson et al. [1999], Zhang et al. [2000], Ames et al. [2006]). The Simulink® (Simulink [2007]) diagram of this model is shown in Fig. 6. The ball velocity is integrated into a position. The corresponding position integrator is initialized with a positive value. This causes the ball to fall and bounce off the ground repeatedly where each collision causes a loss of energy. The collision event is detected by the <= 0 block. Numerically, an event is located immediately (at machine precision) after the ball passes the ground level. Upon location, the output of the <= 0 block becomes true and a '$0To+$' event is generated on the rising trigger reset port of the velocity and position integrator blocks.

When the bounce of the ball is close to the ground, the system is close to the Zeno point, and the simulation will not be able to progress. One way to solve this is to add a '*Stop*' state in the model that is entered when the system is near the Zeno point. The model that exhibits the Zeno behavior is considered *incomplete* and completing it by adding an extra state allows simulation to move beyond the Zeno convergence point (Zheng [2006]). This requires extra effort from the modeler, however, and at times the completing state may be difficult to determine.

Alternatively, to execute the system past the Zeno point, approximate simulation can be used. Several approximate methods exist, such as regularizing the original system (Johansson et al. [1999]).

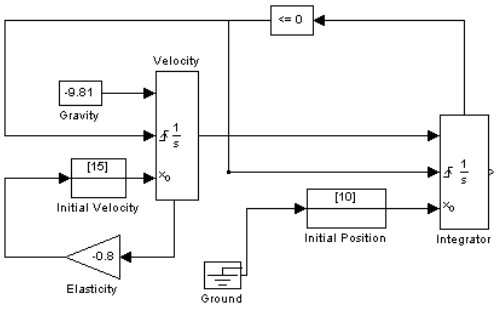


Fig. 6. The bouncing ball model

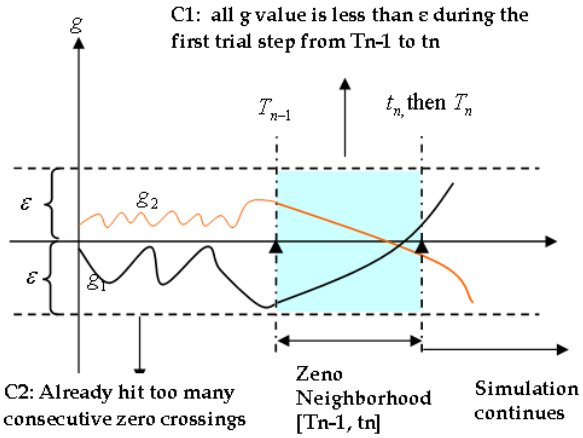


Fig. 7. Dynamic zero crossing location

In this paper, the *dynamic zero crossing location* method is applied. Fig. 7 illustrates how this method dynamically enables and disables zero crossing location when simulating the system in the neighborhood of a Zeno point, and re-initializes the system from that point on. By doing so, the transition time between two events can be adjusted, thereby eliminating the Zeno dynamics. The key elements of this method are: (i) criteria to define a Zeno neighborhood; (ii) how to treat the events detected within the neighborhood of Zeno point; and (iii) what to do after the Zeno point is passed.

To approximate the behavior of the system around the Zeno point, a neighborhood around the Zeno point must first be defined. In this method, two conditions are identified to determine if the system is in the neighborhood of the Zeno point:

(1) At the first trial when the integration advances from T_{n-1} to t_n, if at both ends T_{n-1} and t_n all the zero crossing function values are within a tolerance ϵ.
(2) If before T_{n-1} the simulation already exceeds a certain number of consecutive zero crossings.

If either of these conditions is satisfied, the interval $[T_{n-1}, t_n]$ is defined as a *Zeno neighborhood*.

It is important to note that the first condition only takes effect at the first trial step from T_{n-1} to t_n, but not in the iterative search process. This is because during the search process, when the time interval to locate the event time is reducing to a small tolerance, so do the zero-crossing values at both ends of the interval. These values will be less than ϵ whenever an event is found, even if the system is not a Zeno system. Furthermore, using the first condition only is insufficient because some zero-crossing functions are boolean functions and only return -1 and 1.

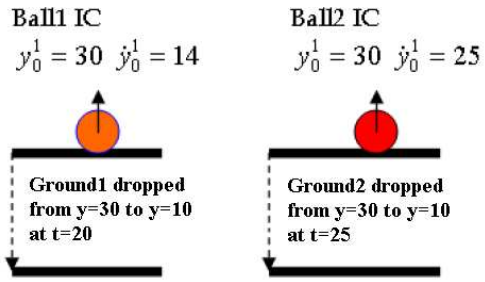


Fig. 8. Double bouncing ball system

Next, if the system is in a Zeno neighborhood, the location of the event will be dynamically disabled. It means the interval $[T_{n-1}, t_n]$ will be used as the final interval at this step. This step is to regulate the transition time and identify the Zeno status of the system. Note the difference between this method and some existing methods that disregard the zero crossings when their magnitude remains below the tolerance level. In the latter case, events occurring in that interval are ignored, which may result in incorrect simulation results.

The next stage accepts the trial integration step t_n as a major step T_n and simulation proceeds by solving for the regular dynamics of the system.

5. BENCHMARK: DOUBLE BOUNCING BALL

A solution strategy was prototyped in Simulink and tested on a benchmark example; a double bouncing ball system. Figure 8 shows the two balls that bounce together. They start from the ground level with different initial vertical velocities up and their ground levels change at different times, comparable to two balls bouncing off a staircase. This system has the following unique features beyond the standard bouncing ball example:

(1) This system experiences several Zeno points at different times, either caused by the dynamics of an individual ball or by the combined dynamics of both balls. This illustrates the robustness of the algorithm to handle multiple zero-crossing functions.
(2) The ground level changes after the system has reached a Zeno point. This illustrates the ability of the algorithm to dynamically enable and disable zero crossing location, depending on the system dynamics.

For this model, if the conventional zero-crossing algorithm is used, the simulation will effectively halt around 14.14 logical seconds, which is when the first ball approaches its Zeno point. No behavior can be generated after this point.

Figure 9 shows a simulation using the method presented in this paper. Several intervals where the designed algorithm is active can be observed. The first one is where the 1^{st} ball has reached a Zeno point and the 2^{nd} ball is still bouncing. This illustrates that for systems with multiple zero-crossing functions, the algorithm individually disables the

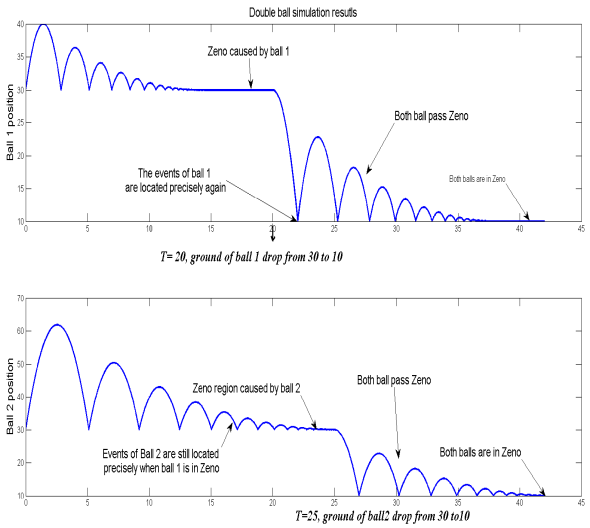


Fig. 9. Simulation results of double bouncing ball model

detection of specific zero-crossing functions. This allows the simulation to progress beyond the first Zeno point while the events of ball 2 are still located accurately. When the ground level of ball 1 is lowered, the ball is moved away from its Zeno point. Simulation of the dynamics proceeds with the events located accurately. The algorithm responds to the change of system dynamics and enables and disables zero crossing location accordingly. At the end of the simulation, both balls have reached their Zeno point with simulation still progressing in logical time.

6. CONCLUSIONS

The generation of behaviors for systems that intersperse continuous-time behavior with discontinuities is notoriously difficult. This paper first studies issues arising from the need to accurately *detect* discrete event from continuous variables exceeding threshold values.

Next, issues arising from the need to accurately *locate* an event are studied, concentrating on discrete events following each other closely in time. Emphasis is put on the problem of discontinuities occurring at an increasing rate which ultimately results in the behavior converging to a limit point in time, so-called *Zeno* behavior.

A method for event detection and location is presented that: (i) prevents detection of the same zero crossing event twice at consecutive time steps, (ii) detects masked even roots zero crossings caused by the presence of multiple zero-crossing functions, and (iii) dynamically enables and disables zero-crossing location to address chattering and Zeno behavior.

Future work aims to concentrate on the analysis of the proposed regularization and to provide error bounds on the numerically generated behaviors.

REFERENCES

Aaron D. Ames, Robert D. Gregg, Haiyang Zheng, and Shankar Sastry. Is there life after Zeno? Taking executions past the breaking (Zeno) point. In *2007 American Control Conference*. Minneapolis, MN, June 14-16, 2006.

Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

François E. Cellier. *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*. PhD diss., ETH, Zurich, Switzerland, 1979.

Joel M. Esposito, Vijay Kumar, and George J. Pappas. Accurate event detection for simulating hybrid systems. *Lecture Notes in Computer Science*, 2034:204–217, 2001.

John Guckenheimer and Stewart Johnson. Planar hybrid systems. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, volume 999, pages 202–225. Springer-Verlag, 1995. Lecture Notes in Computer Science.

Karl H. Johansson, Magnus Egerstedt, John Lygeros, and Shankar Sastry. On the regularization of Zeno hybrid automata. *Systems and Control Letters*, 38:141–150, 1999.

Nancy Lynch and Bruce Krogh, editors. *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

Cleve Moler. Are we there yet? Zero crossing and event handling for differential equations. *EE Times*, pages 16–17, 1997. Simulink 2 Special Edition.

Pieter J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569, pages 164–177. Lecture Notes in Computer Science; Springer-Verlag, 1999.

Pieter J. Mosterman and Gautam Biswas. A hybrid modeling and simulation methodology for dynamic physical systems. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 178(1):5–17, January 2002.

Pieter J. Mosterman, Feng Zhao, and Gautam Biswas. Sliding mode model semantics and simulation for hybrid systems. In Panos Antsaklis, Wolf Kohn, Michael Lemmon, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems V*, pages 218–237. Springer-Verlag, 1999. Lecture Notes in Computer Science.

Taeshin Park and Paul I. Barton. State event location in differential-algebraic models. *ACM Transactions on Modeling and Computer Simulation*, 6(2):137–165, 1996.

Lawrence F. Shampine, Ian Gladwell, and Richard W. Brankin. Reliable solutions of special event location problems for ODEs. *ACM Transactions on Mathematical Software*, 17(1):11–25, 1991.

Simulink. *Using Simulink®*. The MathWorks, Inc., Natick, MA, September 2007.

Frits W. Vaandrager and Jan H. van Schuppen, editors. *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*. Springer-Verlag, March 1999.

Jun Zhang, Karl Henrik Johansson, John Lygeros, and Shankar Sastry. Dynamical systems revisited: Hybrid systems with Zeno executions. In *Hybrid Systems: Computation and Control*, pages 451–464, 2000.

Haiyang Zheng. Simulating Zeno hybrid systems beyond their Zeno points. Technical Report, University of California Berkeley, January 2006.