# Accurate Event Detection for Simulating Hybrid Systems

Joel M. Esposito[1], Vijay Kumar[1], and George J. Pappas[2]

[1] MEAM Department, University of Pennsylvania, Philadelphia, PA 19104,
[2] EE Department, University of Pennsylvania, Philadelphia, PA 19104
{jme,kumar,pappasg}@grip.cis.upenn.edu

**Abstract.** It has been observed that there are a variety of situations in which the most popular hybrid simulation methods can fail to properly detect the occurrence of discrete events. In this paper, we present a method for detecting discrete which, using techniques borrowed from control theory, selects integration step sizes in such a way that the simulation slows down as the state approaches a set which triggers an event (a guard set). Our method guarantees that the state will approach the boundary of this set exponentially; and in the case of linear or polynomial guard descriptions, terminating on it, without entering it. Given that any system with a nonlinear guard description can be transformed to an equivalent system with a linear guard description, this technique is applicable to a broad class of systems. Even in situations where nonlinear guards have not been transformed to the canonical form, the method is still increases the chances of detecting and event in practice. We show how to extend the method to guard sets which are constructed from many simple sets using boolean operators (*e.g.* polyhedral or semi-algebraic sets) . The technique is easily used in combination with existing numerical integration methods and does not adversely affect the underlying accuracy or stability of the algorithms.

## 1 Motivation and Previous Work

Numerical simulation is an important tool for designing and analyzing hybrid systems. In addition to simulation, numerical approximation techniques are increasingly being used in approximate reachability computations, verification and other forms of automated analysis [5], [6], [13]. It is well known that when simulating hybrid systems failure to detect an event can have disastrous results on the global solution due to the discontinuous nature of the problem. Several documents detailing requirements for hybrid simulators list accurate event detection as one primary concern [14], [11].

Figure 1a illustrates graphically the behavior of a generic hybrid system model. At the initial time $t_0$, the mode $q_1$ is active and the continuous system flows according to the differential equation $\dot{x} = f_1(x)$ with initial condition $x_0 = x(t_0)$. Once the condition *Guard* is true the transition from $q_1$ to $q_2$ is enabled; the state may be reset instantaneously and the system enters mode $q_2$ where it then flows according to $\dot{x} = f_2(x)$. The problem we are concerned with

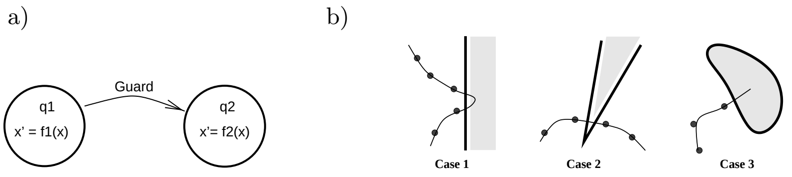

**Fig. 1.** (a) Conceptual model of a generic Hybrid System, (b) three situations for which popular simulators fail to properly detect or localize events.

is correctly detecting the discrete transitions. More formally: problem Given $f : R^n \to R^n$, $x_0 = x(t_0) \in R^n$ and $g : R^n \to R$ such that $g(x_0) < 0$, simulate $\dot{x} = f(x)$, for the time interval $[t_0, t^*]$ where $t^*$ must be computed as the *first* time instant such that $g(x(t)) \geq 0$. problem We assume the guard set has a non-empty interior and is described as $Guard = \{x : g(x) \geq 0\}$ where $g(x)$ is a continuously differentiable. See [12] for an interesting discussion of the unique difficulties associated with solving such problems. It is well known that systems of differential equations with nonlinear guards can be transformed to a equivalent systems with linear guards by appending a new state variable $z = g(x)$ then the new system is

$$\begin{aligned} \dot{x} = f(x) \;&\Leftrightarrow\; \dot{x} = f(x),\; \dot{z} = \frac{\partial g}{\partial x} \cdot f(x) \\ g(x) \geq 0 \quad &\qquad z \geq 0. \end{aligned} \tag{1}$$

Most hybrid system simulators( [1], [9], [18]) divide the task into an event *detection* phase followed by an event *localization* phase. They proceed with the detection phase by checking if $g(x(t_0)) \geq 0$. If the condition is false, numerically integrate the differential equation through one time step, to $t_1 = t_0 + h$ and check if $g(x(t_1)) \geq 0$. This procedure is repeated until a step is taken for which $g(x(t_k)) \geq 0$ is true, at which point an event is assumed to have occured in the interval $(t_{k-1}, t_k]$. Note that the step size $h$ is selected *without* considering the guard dynamics. Some tools then activate a localization phase to determine the time of occurrence more precisely, but some simply assume the event occured at $t_k$. The localization phase is typically a variant on the bisection or bracketing algorithms found in the classical numerical analysis literature. Once the event is localized the integration is stopped, and the transitions occur.

Although this basic technique, first introduced in [4], seems to work well for many problems there are several situations in which it is prone to failure. The situations, discussed below, are illustrated in Figure 1b. The first case is when the trajectory is sufficiently oscullatory that the guard has an even number of roots in the interval $t^* \in (t_k, t_{k+1}]$. A similar situation occurs when the guard set is “thin” or has sharp corners. These two cases are essentially equivalent. Both are situations in which many of the most common *detection* methods can fail. As second class of problems for which the standard technique fails, consider the case when the right hand side of the differential equation is ill-defined for some $x$ such that $g(x) \geq 0$. Perhaps the nature of the system is such that model is only valid in certain regions of the state space. Since the right hand side of the ODE cannot be evaluated at the new point, bisection methods cannot be used to locate the root more precisely. In this situation, almost all common event *localization* methods fail.

Cellier [4] was the first to note that state events warrant special treatment and advocated the discontinuity locking approach still used today. Gear [8] demonstrated the inefficiencies that can result if special techniques are not used. Carver [3] was the first person to notice that the rate of change of the event function along the flow field (*i.e.* the Lie derivative) was a critical quantity in event detection. The idea of differentiating the guard and appending it as an extra state variable to be integrated was introduced there as well. In each of these cases events were detected by simply looking for sign changes in the guard after integrating through one step. As a result they fail to detect an event when there are multiple transitions in a single step. Building on this work, Shampine and his colleagues [12] exploit the fact that interpolation polynomials can be generated for the guard dynamics and are able to correctly identify event occurrences using Strum sequences when the guards are of polynomial expressions but do not use this information to select step sizes. Several similar algorithms for event detection in differential algebraic equations were evaluated in [15]. These techniques are able to detect multiple transition however they tend to be expensive. Most recently, Park and Barton [17] combine some of these ideas and uses methods from interval arithmetic to create efficient tests to determine intervals where it is possible an event had occured. This event detection method seems to be the most reliable technique in the literature, it is streamlined and well suited to stiff problems. However since all of the techniques use the discontinuity locking approach, none of these provides a methodology to select step sizes to ensure that the state never crosses the event surface; thus all fail to localize and event which occurs in the neighborhood of a model singularity.

The idea in this paper is to develop an event detection technique that is not vulnerable to these pitfalls. Using an analogy to control theory we treat the simulated system as a control system, the integration step size as an input, and the guard as the output. The problem is the to select a “feedback law” (a rule for selecting step sizes) such that as the simulation proceeds the system approaches the event surface ($g(x) = 0$) asymptotically, without overshoot ( $g(x) < 0$ always). Since the state approaches the guard asymptotically there is a better chance events are detected and since there is no overshoot there is no risk of crossing a model singularity. In Section 2 we review Linear Multistep numerical integration techniques and introduce the control theoretic concept of input/output linearization which our algorithm is inspired by; in Section 3 we develop in detail the ideas used in the method, culminating in a conceptual algorithm; in Section 4 we successfully solve two example problems which can be problematic for other methods and discuss some of the limitations of the proposed algorithm; finally in Section 5 we summarize our results and comment on future directions.

## 2 Key Concepts

In this section we review numerical integration of ordinary differential equations using Linear Multistep Methods, our prefered integration method. We also introduce the key idea behind our algorithm which draws on the control theoretic concept of input-output linearization.

### 2.1 Review: Numerical Integration with Linear Multistep Methods

Given the system $\dot{x} = f(x, t)$ and $x(0) = x_0$, it is customary to denote the approximate solution at the discrete time $t_k$ as $x_k = x(t_k)$, and then the value of the time derivative may be written as $f_k = f(x_k)$. It is also convention to define the time step as $h_k = t_k - t_{k-1}$. The most general form of a $m$-step linear multistep method (LMSM) is $\sum_{j=0}^{m} \alpha_j x_{k-j+1} = h_k \sum_{j=0}^{m} \beta_j f_{k-j+1}$, where $\alpha_j$ and $\beta_j$ are the coefficients of the method. Particular LMSM's differ in how $\alpha$ and $\beta$ are selected. LMSM's can be broadly divided into two categories: if $\beta_0 = 0$ the method is called *explicit*, otherwise if $\beta_0 \neq 0$ the method is called *implicit*. Although the techniques presented here can be applied to the entire class of explicit LMSM's, the explicit Adams family is by far the most popular and will be used for the purposes of illustration. In such a method, $\alpha_0 = 1$, $\alpha_1 = -1$, and $\alpha_j = 0$ for $j > 1$. The $\beta_j$'s are then selected such that the difference equation

$$x_{k+1} = x_k + h_{k+1} \sum_{j=1}^{m} \beta_j f_{k-j+1}, \tag{2}$$

would *exactly* reproduce the analytical solution $x(t)$ if it were a polynomial of order $m$ or lower. In general the accuracy of the method is proportional to $(h_k)^m$. The Adams family of methods is very popular due to their large region of stability and efficiency. See any numerical analysis text for further details [10]. Often in text books, values of $\beta$ will be supplied as constants; however this is only the case when the step size is constant. In general, $\beta$ is a rational polynomial function of the previous $m$ step sizes, $\beta_j(h_k, \ldots, h_{k-m})$. Multistep methods, as opposed to Runge-Kutta methods, are a natural choice for simulating hybrid systems because the polynomial expressions for $\beta_j$ can be used as interpolants to approximate the solution at off-mesh points.

### 2.2 Feedback Linearization Analogy in Continuous Time

One feature of explicit LMSM's, not present in some other methods, is the fact that $x_{k+1}$ is defined by a difference equation which is affine in the step size $h_k$. This property allows one to draw comparison with nonlinear control systems which often are affine in the input. Following this analogy the difference equation of the numerical method would be the system dynamics, the step size is viewed as the input and the guard function is considered to be the output equation.

For the purposes of illustrating our method, let us imagine for a moment that, instead of belonging to the set of positive integers, we let the step number,

$k$, take on a continuum of values, $k \in [0, \infty)$. Further suppose that $t_k$ is then a continuous function of the real variable $k$, denoted by $t(k)$. Naturally it follows that we would then write $x(t(k))$, and $g(x(t(k)))$. Analogous to the discrete case we then find that the "step size", which is our input variable, can be viewed as $h(k) = \frac{dt}{dk}$. The dynamics of the event function (our output function) are then

$$\frac{dg}{dk} = \left( \frac{\partial g}{\partial x} \frac{dx}{dt} \right) \frac{dt}{dk}, \tag{3}$$

since by definition $\frac{dx}{dt} = f(x)$ this can be rewritten as,

$$\frac{dg}{dk} = (L_f g)\, h(k). \tag{4}$$

Note that the Lie derivative, $L_f g = \frac{\partial g}{\partial x} \cdot f$, has a geometric interpretation here as the time derivative of $g(x)$ along trajectories of the ODE.

We would like to select $h(k)$ in such a way as to ensure that $g(x) \to 0$ as $k \to \infty$. This may be accomplished by a technique from nonlinear control theory called feedback linearization (see for example [2]). Assuming the Lie derivative is non-zero, selecting

$$h(k) = -\gamma \frac{g(x)}{L_f g}, \tag{5}$$

and substituting into eq.(4) results in

$$\frac{dg}{dk} = -\gamma g \tag{6}$$

where $\gamma$ is some positive constant to be selected by the user. The solution to the ODE is then $g(k) = g(0) \exp^{-\gamma \cdot k}$; which implies $g(k) \to 0$ exponentially, as $k \to \infty$. Thus, by judicious selection of the input, one may cancel the nonlinearities and stabilize the guard dynamics. In terms of simulation, by selecting the step size appropriately using eq.(5) we are able to re-parameterize time in order to make the guard (as a function of the step number) behave as a linear differential equation which has a stable equilibrium point on the surface $g(x) = 0$.

## 3 Simulation Algorithm

In this section we describe the ideas used in our simulation algorithm: methods for computing step sizes depending on the form of the guards (Sect. 3.1– 3.3), computation of candidate step sizes (Sect. 3.4), dealing with boolean combinations of guards (Sect. 3.5), merging the candidate step size for event detection with the ideal step sizes computed for integration accuracy and other implementation details (Sect. 3.6 and 3.7). Finally, in Section 3.8 these ideas are presented as a concrete algorithm.

While Sect. 2.2 contains a useful way of thinking of such systems, the simulated system evolves in discrete time. For a linear multistep method the dynamics are

$$x(t_k + h_{k+1}) = x_{k+1} = x_k + h_{k+1}\{\sum_{j=1}^{m} \beta_j f_{k-j+1}\} \tag{7}$$

which implies the guard dynamics are

$$g(x(t_k + h_{k+1})) = g_{k+1} = g(\, x_k + h_{k+1}\{\sum_{j=1}^{m} \beta_j f_{k-j+1}\}\, ). \tag{8}$$

Selecting $h_{k+1}$ to produce the desired behavior is somewhat more difficult in discrete time.

## 3.1 Symbolic Inverse

In theory, provided the guard is an invertible function (with respect to time along a given integral curve), we can select

$$h_{k+1} = \frac{-x_k + g^{-1}(\gamma g(x_k))}{\bar{f}_\beta} \tag{9}$$

where the vector $\bar{f}_\beta = \sum_{j=1}^{m} \beta_j f_{k-j+1}$, yielding the *difference* equation $g_{k+1} = \gamma g_k$, which has the solution $g_k = g_0 \gamma^k$ and converges exponentially to $g = 0$ provided $0 \leq \gamma < 1$. This naturally assumes one can compute the symbolic inverse of the guard, $g^{-1}(h_{k+1})$, which is an unrealistic assumption in practice.

## 3.2 Exact Linearization

While it is unlikely that one would have a symbolic expression for the inverse of $g(x(t))$, exact linearization is possible for all guards with Taylor series expansions of finite length (i.e polynomial or linear guards). We illustrate this idea with linear guards, since they can be used to model a wide class of systems either through approximation or by transforming nonlinear guards to linear ones using eq.(1). If our event function is of the form $g(x) = a \cdot x + b$, where $a \in R^n$ and $b \in R$ are constant eq.(4) becomes

$$g_{k+1}(h_{k+1}) = g_k + h_{k+1} \frac{\partial g}{\partial x} \bar{f}_\beta \tag{10}$$

which is essentially a Taylor series expansion in $h_{k+1}$ about $x_k$. Since $\frac{\partial g}{\partial x} \bar{f}_\beta$ is simply the Lie derivative $L_{\bar{f}_\beta} g$, select

$$h_{k+1} = \frac{(\gamma - 1) g_k}{L_{\bar{f}_\beta} g}. \tag{11}$$

Polynomial guards can be handled in a similar manner, by calculating and inverting their Taylor series expansions in $h_{k+1}$.

## 3.3 Approximate Linearization

If nonlinear guards with a Taylor series expansion of infinite length are not transformed to linear guards, an approximate linearization technique can be used. Approximations using a Taylor series expansion gives

$$g_{k+1}(h_{k+1}) = g(x_k) + L_{\bar{f}_\beta} g h_{k+1} + \frac{1}{2} L^2_{\bar{f}_\beta} g h^2_{k+1} + \dots \tag{12}$$

It is possible to compute the inverse of $g$ as a function of $h$ for the Taylor series expansion using a result due to Grobner often referred to as the Lie series

$$g^{-1}(h) = \sum_{p=0}^{\infty} \frac{1}{p!} \{ \frac{1}{\frac{\partial g}{\partial x} \cdot f} \frac{\partial}{\partial h} \}^p h \|_{x=x_k, h=0} \cdot [(\gamma - 1) g_k]^p. \tag{13}$$

While the result is defined as an infinite series, a finite number of terms can be used to compute an approximate linearization. One sided convergence is no longer guaranteed since uncanceled terms act as forcing functions, but by selecting a small value of $\gamma$ the state still approaches the event surface slowly, increasing the likelihood that the event will be detected. This method seems to work well in practice since $h$ is typically small implying that the higher order terms are usually correspondingly small

## 3.4 Computation of Step Sizes

As mentioned earlier, the $\beta$'s for the Adams Method are only constant in the special case of constant step size. Since we are proposing to adjust the step size dynamically, the $\beta$'s in the above discussion are not constant, but rather are rational polynomial functions of $h_{k+1}$. Computing the correct step size with eq.(11), for example, then entails finding the roots of a polynomial in $h_{k+1}$. For example in the case of two step Adams method $\beta_1 = (2h_k)/h_{k+1}$ and $\beta_2 = 1 - (2h_k)/h_{k+1}$. Substituting the expressions for $\beta$ into eq.(11) and rearranging gives

$$z = Roots[a h^2_{k+1} + b h_{k+1} + c] \tag{14}$$

where $a = 1/2 \cdot h_k[\partial g/\partial x \cdot (f_k + f_{k-1})]$, $b = \partial g/\partial x \cdot f_k$ and $c = -(\gamma - 1)g(x_k)$. Eq.(14) must be solved for $h_{k+1}$ at every time step. Similar polynomials can be constructed using eq.(9) or eq.(13). Various algorithms for computing the roots of polynomials exist, most involve constructing the companion matrix and computing its eigenvalues. In general the polynomial equation determining $h_{k+1}$ will have $m$ roots (for an m-step multistep method), however only positive real roots should be considered as candidates for event times, since negative roots correspond to past events, while complex roots are physically meaningless. Assume the positive real roots have been ordered from smallest to largest $\{r_1, r_2, \dots, r_p\} \subset z$, then in the simplest case of a single guard, $r_1$ corresponds to the first event and hence is the proper choice for $h_{k+1}$. If there are no positive real roots set $h_{k+1} = \infty$.

## 3.5 Boolean Combinations of Guards

In many realistic system models, complex guards may be composed of several algebraic inequalities joined or modified by boolean operators (*e.g.* polyhedrals or semi-algebraic sets). If the guard is $(g^a(x) \geq 0) \bigvee (g^b(x) \geq 0)$, the situation is accommodated by computing $r_1^a$ and $r_1^b$, the smallest positive real roots for eq.(14) using $g^a(x)$ and $g^b(x)$, and selecting $h_{k+1} = \min[r_1^a, r_1^b]$.

In the case of $(g^a(x) \geq 0) \bigwedge (g^b(x) \geq 0)$, we compute at time $t_k$ the sets of positive ordered real roots $\{r_1^a, r_2^a, \ldots, \}$ and $\{r_1^b, r_2^b, \ldots, \}$ using eq.(14). Then

1. if $g^a(x_k) < 0$ but $g^b(x_k) \geq 0$; and if $r_1^a < r_1^b$, let $h_{k+1} = r_1^a$.
2. if $g^b(x_k) < 0$ but $g^a(x_k) \geq 0$; and if $r_1^b < r_1^a$, let $h_{k+1} = r_1^b$.
3. if both $g^a(x) < 0$ and $g^b(x) < 0$; and if *either* $r_1^a < r_1^b < r_2^a$ *or* $r_1^b < r_1^a < r_2^b$; let $h_{k+1} = r_2^b$ or $r_2^a$ respectively.

Guards prefaced with a $\neg$ operator can be converted to the standard form by changing their sign, that is by using $-g(x) \geq 0$ rather than $g(x) \geq 0$.

## 3.6 Final Selection

In practice, event considerations are not the only criteria which determine the appropriate step size to be used in simulation. Often the simulation will specify some minimum step size, $h_{\min}$ , below which roundoff errors affect the stability of the computation. In addition, most modern numerical integrators estimate an ideal step size based on truncation error considerations, $h_{err}$. The resulting step size selected by our algorithm based on event detection, $h_{k+1}$, can be easily incorporated into existing integration algorithms by selecting the actual step size as

$$h = \max[h_{\min},\ \min(h_{k+1}, h_{err})\,]. \tag{15}$$

In this way the original accuracy and stability properties of the integration algorithm are preserved.

## 3.7 Termination Criteria

In cases where the guards have a Taylor series expansion of finite length, $\gamma = 0$ will yield exact and rapid convergence to the event surface; therefore the algorithm should be terminated when $g(x_{k+1}) = 0$ If the guards are more general nonlinear functions , exact convergence is not guaranteed. In such situations, conservatively selecting $0 < \gamma < 1$ will cause the simulator to take successively smaller steps toward the surface. However, selecting $\gamma$ too large results in slow convergence rate and a very small $\gamma$ can risk overshooting the guard, in practice we have found $0.05 < \gamma < 0.5$ to be a good selection. Slowing down the simulation in this manner has the effect of dramatically increasing the chances an event will be properly detected and may event be useful when exact linearization is possible. Since steps are taken in such a way that the value of the guard approaches zero asymptotically, it may take an infinite number of steps to reach zero exactly. Therefore the user must set a small threshold $\epsilon \geq 0$ such that the procedure is terminated when $g(x) \geq -\epsilon$. Alternatively one could choose to stop the procedure once the computed time step is smaller than $h_{min}$.

## 3.8 Algorithm

All of these ideas are assembled into an algorithm and implemented in Matlab.
**Given** by the user upon initialization:

– A set of atomic propositions of the form $g^a(x) \leq 0$, $g^b(x) \leq 0$, $g^c(x) \leq 0$, $\ldots$ joined or modified using the operators $\bigvee$, $\bigwedge$, and $\neg$.
– the gain, $0 \leq \gamma < 1$; and termination tolerance $\epsilon \geq 0$.

**Preprocessing**

1. convert any guards of the form $\neg g(x) \leq 0$ to $-g(x) \leq 0$.
2. *if desired*, convert any nonlinear guards to linear guards, using the transformation described in eq.(1), by appending an extra state variable.

**Repeat until termination**
**Get** from the integration algorithm at each iteration:

– $m$ previous derivatives used in the multistep integration method, $f_k, f_{k-1}$, $\cdots$ , $f_{k-m}$
– ideal step size for controlling the truncation error,$h_{err}$ and minimum allowable step size, $h_{min}$

**Main Algorithm**

1. for each atomic proposition $g^a$, $g^b$, $\ldots$, $g^i$, $\ldots$ compute a candidate step size using the appropriate method:
   a) *symbolic inverse* $o(g^i(x))^{-1}$ *given by user* –
$$\textit{Roots}\ [h\bar{f}_\beta(h) + x_k - (g^i)^{-1}(\gamma g^i(x_k))] = z^i$$
   b) $g^i(x)$ *is linear* or has been converted to linear form and $L_{\bar{f}_\beta} g \neq 0$ –
$$\textit{Roots}\ [hL_{\bar{f}_\beta(h)} g^i - (\gamma - 1)g^i(x_k)] = z^i$$
   c) $g^i(x)$ *is a polynomial of order* $N$ –
$$\textit{Roots}\ [\sum_{p=1}^{N} L^p_{\bar{f}_\beta(h)} g^i \frac{h^p}{p!} - (\gamma - 1)g^i(x_k)] = z^i$$
   d) *nonlinear* $g^i(x)$ – compute roots, $z^i$, using Lie series (eq. 13).
2. for each set of roots from the previous step $z^a$, $z^b$, etc. discard any negative or complex roots. If there are no positive real roots for a given $z^i$ set $h^i = \infty$; otherwise sort the positive real roots in ascending order $r^i = \{r_1, r_2, \ldots\}$.
3. Using $r^a, r^b, \ldots,$ recursively compute a composite step size,$r^*$ for each boolean conjunction using the rules in section 3.5.
4. combine this result with the step size computed in the integration algorithm using $h = \max[h_{min}, \min(r^*, h_{err})]$
5. integrate through one step of size $h$. If $g(x_{k+1}) \geq -\epsilon$ terminate; else, repeat

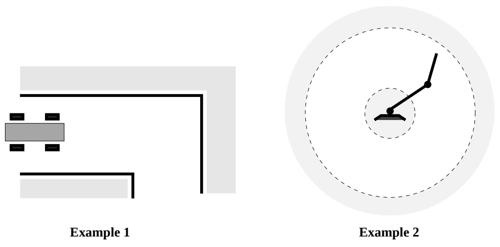

**Fig. 2.** Two examples: (1) an autonomous robot navigating a corridor; (2) a planar two link manipulator with workspace limitations.

## 4 Examples and Discussion

In this section we illustrate the effectiveness of our algorithm using the two examples shown in Figure 2. The first, controlling a car-like robot, represents a situation in which other event detection methods fail, because the guard set possesses "sharp" corners. The second, a planar manipulator with workspace limitations, illustrates a situation in which many event localization methods fail due to a model singularity. We also discuss some shortcomings of the proposed algorithm.

*Example 1.* Consider the nonholonomic cart trying to navigate an indoor environment as shown in Figure 2. The kinematic equations are

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{16}$$

where the inputs $u_1$ and $u_2$ are the forward velocity and turning rate. The details of the robot control problem and the history of $u_1$ and $u_2$ are omitted here, but it is assumed to be provided by a controller. The goal here is to verify the efficacy of the controller and in particular, to verify that the robot does not collide with the obstacles. For the sake of simplicity, we ignore the physical size of the robot and simply think of it as a point. Thus the guard(s) for the simulation are given by the equations of the walls

$$((y - 0.5 \geq 0) \bigvee (x - 3.5 \geq 0)) \bigvee ((-y - 0.4 \geq 0) \bigwedge (2.8 - x \geq 0)). \tag{17}$$

Figure 3a displays a situation for which the standard algorithm fails. Integration points are computed which happen to land just outside the guard region. Thus the simulator detects no collision when in fact the robot has collided with the walls, near the corner ($x = 2.8$, $y = -0.4$). Figure 3b illustrates the method presented in this paper. Observe how the integrator slows down as it approaches the event surface. Note that in this example the gain was selected in such a way as to produce a very gradual slow down, for the purposes of illustrating the technique. In practice, since the guards are linear, a gain of $\gamma = 0$ could have been used to force fast convergence.

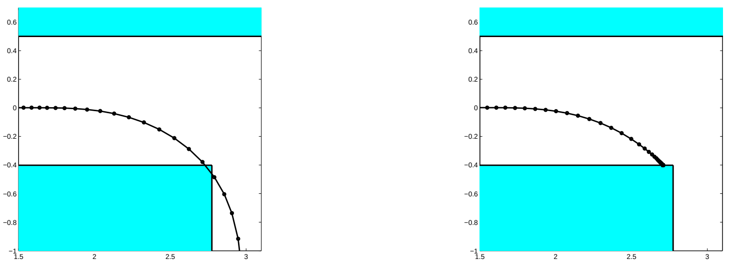

**Fig. 3.** Simulations of the mobile robot in example 1: (a) standard simulation technique fails to detect the collision; (b) our method slows down as it approaches the event surface.

*Example 2.* Consider the planar two link manipulator, as shown in Figure 2, with the kinematic equations

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \tag{18}$$

desired $(x, y)$ positions for the end point are fed to the controller from a high level planner and the model is required to calculate $\theta_1$ and $\theta_2$ to achieve these positions. If the length of the proximal link is $l_1$ and the distal link is $l_2$, the appropriate inverse kinematics relation to compute $\theta_1, \theta_2$ as a function of $(x, y)$ are

$$\theta_1 = \arctan 2\left[\frac{-y}{\sqrt{x^2+y^2}}, \frac{-x}{\sqrt{x^2+y^2}}\right] \pm \cos^{-1}\left[\frac{-(x^2+y^2+l_1^2-l_2^2)}{2l_1\sqrt{x^2+y^2}}\right] \tag{19}$$

$$\theta_2 = \arctan 2\left[\frac{y - l_1\sin(\theta_1)}{l_2}, \frac{x - l_1\cos(\theta_1)}{l_2}\right] - \theta_1 \tag{20}$$

Note that it is possible for the high level planner to be unaware of the specifics of the manipulator and specify $(x, y)$ points which are outside the set of reachable positions of the manipulator, in such cases the arguments of the $\cos^{-1}$ function would fall outside of the range of $[-1, 1]$ and the right hand side of the differential equation becomes ill-defined. In this case, given $l_1 > l_2$ the guard would be

$$(\sqrt{(x^2+y^2)} \leq (l_1+l_2)) \bigwedge (\sqrt{(x^2+y^2)} \geq (l_1-l_2)) \tag{21}$$

with $x = l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2)$,$y = l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2)$.

Figure 4a displays a simulation of the two link manipulator attempting to track a reference trajectory, which is a straight line in Cartesian space. In this case the reference trajectory eventually falls outside the workspace of the manipulator, where the right hand side of the differential equation becomes complex. The traditional integrator generates a point near the edge of the workspace and its next point falls outside the workspace. Because the vector field is ill-defined there, it is unable to correctly compute this new point, nor is it able to activate

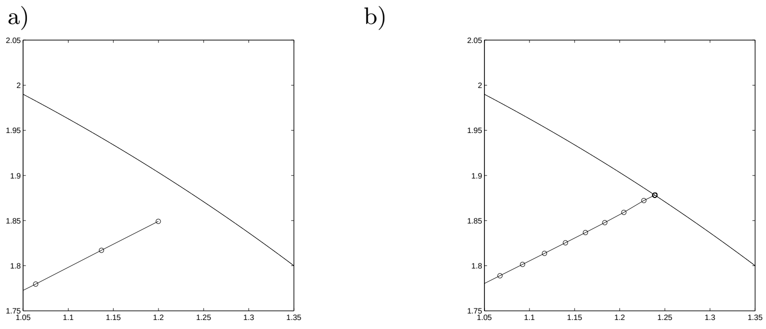

**Fig. 4.** Simulations of the two link manipulator from example 2: (a) root bracketing methods cannot be used since the vector field is ill-defined out side the workspace; (b) our method approaches the surface asymptotically without every requiring a function evaluation outside the workspace.

its root finding algorithm (bracketing technique) since it requires an initial point on each side of the guard. The output of our algorithm is shown in Figure 4b. Successively smaller steps are taken as the state approaches the boundary of the workspace.

*Discussion.* It should be said that, although our method is capable of terminating the simulation at $t_k$ such that $g(x_k) = 0$ exactly, in some situations, or coming arbitrarily close to it in others, it can only be considered accurate insofar as the underlying integration method accurately reproduces the exact solution to the differential equation. That is to say that while $g(x_k)$ will equal zero exactly, $x_k$ itself is not exact since it is generated through an approximation algorithm, as in all numerical analysis. Other limitations include:

– In eq.(11), which determines the step size, one must divide by the quantity $L_{\bar{f}_\beta} g$. Obviously the method is not applicable when this quantity is zero. Infact, by the inverse function theorem, $L_{\bar{f}_\beta} g = 0$ implies that the inverse of $g(t)$ used in eq.(9) does not exist. Geometrically, the differential equation is flowing purely tangential to the boundary of the guard set, an alternative method is required.
– The method requires solving for roots of eq.(14) at every step, despite the fact that that specialized algorithms exist, this computation can be a bit time consuming for higher order methods (higher order polynomials). We feel that given the importance of discrete event detection in accurate simulation this additional effort is worthwhile although an efficient exclusion test would improve the performance.

## 5 Conclusions and Future Work

It has been observed that there are a variety of situations in which one of the most popular hybrid simulation methods can fail to properly detect or localize the occurrence of discrete events: either due to a multiple number of zero crossings within a single step or because of model singularities. We present a method for detecting discrete events which, using techniques borrowed from control theory, selects integration step sizes in such a way that the simulation slows down as it approaches a guard. Our method guarantees that the simulation will land *exactly* on the event surface for any guard which has a Taylor series expansion of finite length. Given that any nonlinear guard can be transformed to a linear form, this technique is applicable to a broad class of systems. Even in situations where nonlinear guards have not been transformed to the canonical form, the method is still quite useful in practice. We show how to extend the method to complex guards which are built up from many simple algebraic inequalities using the boolean operators *and*, *or* and *not*. In this way polyhedral or semi-algebraic guards sets can be handled. The technique is easily used in combination with existing integration algorithms and does not adversely affect the underlying accuracy or stability of the numerical integration technique. Ultimately the framework presented here will be coded in Java (presently written in Matlab) and incorporated into the CHARON [16] simulation suite.

While our method requires a variable step size integration method, it has been observed that when simulating large systems such as Automated Highway Systems with 1000+ vehicles, traditional variable step size schemes are unacceptable since they require all components to be simulated at the same rate. Thus if only two of the vehicles actually necessitate a step size reduction, the entire system must be slowed down to the smallest common step size, creating gross inefficiencies. To address this problem, we are currently considering using the techniques presented here in conjunction with *multirate integration methods* such as those presented in [7]. When integrating a systems of ODEs, multirate methods use a different step size for each component. Thus, when a particular component of the set of equations is changing rapidly a small step size may be used without unnecessarily slowing down the integration rate for other slowly changing components. Multirate implementation would prevent agents not involved in the event from being simulated at an unnecessarily slow rate. We believe that these two techniques complement each other and can be used to develop a powerful simulation tool for multiagent and hierarchical hybrid systems.

**Acknowledgments.** We gratefully acknowledge support from DARPA grant MOBIES F33615-00-C1707 and NSF CDS-97-03220. The first author is also partially supported by a DoE GAANN grant.

## References

1. A. Gollu A. Deshpande and L. Semenzato. Shift programming language and run-time system for dynamic networks of hybrid automata. *California PATH*, 1995.
2. A.Isidori. *Nonlinear Control Systems.* Springer, London, 1995.
3. M.B. Carver. Efficeint integration over discontinuities in ordinary differential equation simulations. *Mathematics and Computers in Simulation*, XX:190–196, 1978.

4. F. Cellier. *Combined discrete/ continuous system simulation by use of digital computers: techniques and tools.* PhD thesis, ETH Zurich, Zurich, Switzerland, 1979.
5. A. Chutinam and B. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In F. Vaandrager and J. H. van Schuppen, editors, *Hybrid Systems : Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.
6. T. Dang and O. Maler. Reachability analysis via face lifting. In T. Henzinger and S. Sastry, editors, *Hybrid Systems : Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*, pages 96–109. Springer Verlag, Berlin, 1998.
7. J. Esposito and V. Kumar. Efficient dynamic simulation of robotic systems with hierarchy. submitted to International Conference on Robotics and Automation 2001.
8. C.W. Gear and O.Osterby. Solving ordinary differential equations with discontinuities. Technical report, Dept. of Comput. Sci., University of Illinois, 1981.
9. D. Bruck H. Elmqvist and M. Otter. Dymola – user's manual. Dynasim AB Research Park Ideon, Lund Switzerland, 1996.
10. S. Campbell K.Benan and L. Petzold. *Numerical solutions of initial value problems.* North Holland, London, 1989.
11. S. Kowaleski, M.Fritz, H. Graf, J.Preubig, S.Simon, O.Stursberg, and H.Treseler. A case study in tool-aided analysis of discretely controled continuous systems: the two tanks problem. In *Hybrid Systems V*, Lecture Notes in Computer Science. Springer Verlag, 1998.
12. L.F.Shampine, I.Gladwell, and R.W.Brankin. Reliable solution of special event location problems for ODEs. *ACM transactions on Mathematical Software*, 17(1):11–25, March 1991.
13. Ian Mitchell and Claire Tomlin. Level set methods for computation in hybrid systems. In N. Lynch and B. H. Krogh, editors, *Hybrid Systems : Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 310–323. Springer Verlag, 2000.
14. P.Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In F.W. Vaandrager and J. H. van Schuppen, editors, *Hybrid Systems : Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 163–177. Springer Verlag, 1999.
15. A.J. Preston and M.Berzins. Algorithms for the location of discontinuities in dynamic simualtion problems. *Computers in Chemical Engineering*, 15(10):701–713, 1991.
16. R.Alur, R. Grosse, Y.Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in charon. *Hybrid Systems Computation and Control: Third international workshop*, 3:6–19, 2000.
17. T.Park and P.Barton. State event location in differential-algebraic models. *ACM transactions on modeling and computer simulation*, 6(2):137–165, 1996.
18. J.Liu X.Lui, T.J.Koo, B.Sinopoli, S.Sastry, and E.A.Lee. A hierarchical hybrid system model and its simulation. *Proceedings of the* 38th *Conference on Decision and Control*, pages 2407–2411, 1999.