An Asynchronous Integration and Event Detection Algorithm for Simulating Multi-Agent Hybrid Systems

JOEL M. ESPOSITO
United States Naval Academy
and
VIJAY KUMAR
University of Pennsylvania

A simulation algorithm is presented for multi-agent hybrid systems—systems consisting of many sets of nonsmooth differential equations—such as systems involving multiple rigid bodies, vehicles, or airplanes. The differential equations are partitioned into coupled subsystems, called "agents"; and the conditions which trigger the discontinuities in the derivatives, called "events", may depend on the global state vector. Such systems normally require significant computational resources to simulate because a global time step is used to ensure the discontinuity is properly handled. When the number of systems is large, forcing all system to be simulated at the same rate creates a computational bottleneck, dramatically decreasing efficiency. By using a control systems approach for selecting integration step sizes, we avoid using a global time step. Each subsystem can be simulated asynchronously when the state is away from the event. As the state approaches the event, the simulation is able to synchronize each of the local time clocks in such a way that the discontinuities are properly handled without the need for "roll back". The algorithm's operation and utility is demonstrated on an example problem inspired by autonomous highway vehicles. Using a combination of stochastic modelling and numerical experiments we show that the algorithm requires significantly less computation time when compared with traditional simulation techniques for such problems, and scales more favorably with problem size.

Categories and Subject Descriptors: I.6.8 [**Simulation and Modeling**]: Types of Simulation

General Terms: Algorithms

Additional Key Words and Phrases: Event detection, hybrid systems, multi-agent systems, numerical integration

Authors' addresses: J. M. Esposito, U. S. Naval Academy, Weapons & Systems Engineering, 121 Blake Road, Annapolis, MD 21402-5000; email: esposito@usna.edu; V. Kumar, Department of Mechanical Engineering & Applied Mechanics, University of Pennsylvania, Room 111, Towne Building, 220 South 33rd street, Philadelphia, PA 19104-6315; email: Kumar@central.cis.upenn.edu.

1. INTRODUCTION

Many engineering systems are best described by sets of ordinary differential equations (ODEs) with discontinuous right-hand sides. Examples include phase transitions, contact mechanics, or the dynamics of physical systems controlled by digital computers. Such systems arise in many contexts and are often referred to as *hybrid systems*, switched systems, or nonsmooth systems. They can also be though of as discrete event systems augmented with differential equations. See Branicky [1995] for a discussion of various modeling paradigms. The most basic form of such a systems is

$$\dot{x} = \begin{cases} f_a(x,t), & g(x) < 0 \\ f_b(x,t), & g(x) \geq 0. \end{cases} \tag{1}$$

Each of the functions f_a and f_b describing the derivative represent distinct modes of operation of the hybrid system. We will refer to each function simply as a mode.

Changes from one mode of operation to another, called mode switches, are caused by state events or simply *events*. The occurrence of these events is triggered by the zeros of an event function $g(x(t))$ (also called a guard or discontinuity function in the literature). When such an event occurs the first derivative of the solution trajectory becomes discontinuous. Of particular interest is the fact that embedded or software based control systems can be modeled in such a fashion (see, e.g., Maler and Pnueli [2003] and other in that series). In such a case, there can be many modes and their connectivity may be quite complex.

Multi-agent hybrid systems are groups of individual interacting hybrid systems. More precisely, we define multi-agent hybrid systems as collections of individual hybrid systems, each called agents, in which the there is no agent-to-agent coupling in the right hand sides of the differential equations. However, the event functions which trigger mode transitions may depend on the global state vector. This concept is best illustrated through an example. Consider the simplified kinematics of two automated highway vehicles:

$$\begin{bmatrix} \dot{z^1} \\ \dot{y^1} \\ \dot{\theta^1} \end{bmatrix} = \begin{bmatrix} \cos(\theta^1) \\ \sin(\theta^1) \\ 0 \end{bmatrix} v^1(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega^1(t), \tag{2}$$

$$\begin{bmatrix} \dot{z^2} \\ \dot{y^2} \\ \dot{\theta^2} \end{bmatrix} = \begin{bmatrix} \cos(\theta^2) \\ \sin(\theta^2) \\ 0 \end{bmatrix} v^2(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega^2(t), \tag{3}$$

where the superscripts 1 and 2 are used to distinguish between the state (position and orientation in the plane) of the first and second cars (i.e., agents 1 and 2), $x^1 = [z^1 \ y^1 \ \theta^1]$ and $x^2 = [z^2 \ y^2 \ \theta^2]$. The functions $v^1(t)$, $v^2(t)$, $\omega^1(t)$ and $\omega^2(t)$ are the forward and turning velocities of the two cars respectively. These time-dependent (perhaps implicitly) functions are considered inputs to the system and are supplied by the automatic control system. The control system may have

a library of possible functions for the velocity and turning rate, which it selects among based on current operating conditions. For example, under nominal operation the vehicles are to drive in a straight line; however, if the two vehicles come within R feet of each other, an emergency collision avoidance controller would be activated and both the cars would steer away from one another. In such a situation perhaps:

$$\omega^1 = \begin{cases} 0, & g(x^1, x^2) \geq 0 \\ 1, & g(x^1, x^2) < 0 \end{cases} \tag{4}$$

$$g(x^1, x^2) = (x^1 - x^2)^2 + (y^1 - y^2)^2 - R^2 \leq 0. \tag{5}$$

Obviously much more sophisticated examples and control laws could be used, however, the representative features of this example are that the right-hand sides of the differential equations for an individual car's dynamics do not depend on the state of the other car; yet the condition for switching control laws depends on the states of several agents.

Examples of such systems abound. Automated highway systems [Deshpande and Semenzato 1995], free-flight air traffic control [Tomlin et al. 1998], cooperative mobile robotic systems [Fierro et al. 2002], cellular biology [Alur et al. 2001], and multibody systems simulated for graphics applications all can be modeled this way—the dynamics of each vehicle, plane, or robot are decoupled; however, certain critical events which are relevant to the simulation (e.g., collisions) depend on the states of pairs of agents. Also, note that the partitioning of the agents can change throughout the course of the simulation. Two bodies can collide and stick together for a brief duration, in which case their ODE's are coupled and they are a single agent. Likewise, the collision avoidance mode for the automated vehicle example may make use of a closed loop feedback law, in which case the velocity for car 1 is determined based on the position of car 2. Coupling of the differential equations, either through software or physical interaction, makes it necessary to consider the two systems as subsystems of a single agent for the purposes of simulation.

Traditionally, the need to properly detect and manage such state events has suggested the use of a single global integration step size, which is necessarily the minimum acceptable step size across all agents. Because the differential equations of each agent are decoupled, this is not required and often leads to severe loss of simulation efficiency, especially in applications with a large number of agents. This phenomena is explained in Section 2.

This article introduces an alternate simulation algorithm that allows each agent to use a different step size, resulting in an asynchronous simulation. The effect is that the average step size is much larger resulting in significant decreases in computation time. In Section 3, we review work related to asynchronous simulation. In Section 4, our methodology for selecting integration step sizes, while properly handling state events, is presented along with results on the need for "roll back". The overall algorithm is introduced in Section 5 and illustrated via example in Section 6. Section 7 quantifies the algorithms advantages over the traditional approach in terms of computational cost. The role of the step size selection scheme is modelled statistically and the effect of

the additional overhead of the new algorithm is examined. In Section 8, the performance improvement is determined experimentally.

2. BACKGROUND AND CHALLENGES

In Cellier [1979], it was shown that the proper way to simulate any hybrid system is to numerically integrate all of the differential equations until t^* – the *first* time at which $g(x(t)) = 0$. At this point, the numerical integration is stopped, any applicable mode switches are activated and the integration may be restarted, using $x(t^*)$ as the new initial condition. This technique is referred to as *discontinuity locking* and is widely accepted as the standard hybrid system simulation methodology. This requirement of stopping the integration precisely when $g(x(t^*)) = 0$ gives rise to the *event detection problem*. Since numerical integrations are performed in discrete time, $T \in \{t_1, t_2, \ldots, t_k, \ldots\}$ with step sizes $h_k = t_k - t_{k-1}$, it is difficult to find t^* exactly. Much work has been done on the problem and most reliable approaches use interpolants to approximate the state between steps and then check these interpolants to find the time of zero crossings of $g(x)$ (e.g., Shampine et al. [1991], Park and Barton [1996], Esposito et al. [2001b], and Bahl and Linninger [2001]).

It is well known that, when simulating hybrid systems, a failure to detect an event can have disastrous results on the global solution due to the nonsmooth nature of the problem [Branicky 1995]. Works detailing requirements for hybrid simulators list accurate event detection as a primary concern [Mosterman 1999]. Event detection in hybrid systems is, in itself, a very difficult problem [Shampine et al. 1991].

A second important issue in numerical integration is managing the tradeoff between efficiency and the desired accuracy, on-the-fly, by varying the step size. Large step sizes result in more efficient simulations but decrease the accuracy and stability of the results. Small step sizes are required to maintain stringent accuracy requirements when the solution to the differential equation is ill-behaved, however the computations become extremely expensive due to the large number of steps needed to complete the simulation. Automatic step size selection schemes attempt to optimize this tradeoff by selecting the largest possible step size consistent three criteria: (1) maintaining the estimated truncation error within a desired range; (2) maintaining numerical stability; and (3) accurate event detection.

Traditionally, all hybrid system simulators use a single global notion of time, meaning the integration of all agents is synchronized using the same time discretization. In such a scheme, at each iteration, a candidate integration step size, h^i for agent i, is computed for each agent. This candidate represents the maximum step which will result in an acceptable truncation error. The only acceptable choice of a global step size–if one wishes to maintain each agent with the acceptable range of truncation error–is the *minimum* among all the agent's candidate step sizes, $h = \min[h^1, \ldots, h^N]$. This selection means some agents will be simulated with an unnaturally small step size—resulting in a generally inefficient integration scheme. In particular as the number of agents increases, h tends to decrease on average.

Note that the decision to use a global step size is *not* motivated by the differential equations. Had this been a purely smooth system of decoupled differential equations (no events), the optimal efficiency would be obtained by permitting each agent to use the largest acceptable step size. This would result in an asynchronous simulation in which each agent proceeded at its own "fastest" acceptable simulation rate.

The motivation for using a traditional global choice of step sizes is entirely attributed to the coupled nature of the event detection problem. If agents 1 and 2 are simulated with different step sizes, and hence different time meshes $T^1 = \{t_1^1, t_2^1, \ldots, t_k^1\}$ and $T^2 = \{t_1^2, t_2^2, \ldots, t_k^2\}$, state estimates for the two agents are produced at different points in time. Deciding if an event has occurred now becomes nontrivial because merely evaluating $g(x^1, x^2) \leq 0$ requires x^1 and x^2 be evaluated at the same time instant—that is, $g(x^1(t_k), x^2(t_k))$ is meaningful while $g(x^1(t_j), x^2(t_k))$ with $t_j \neq t_k$ is not. Rather than address this issue traditional simulators simply use a global notion of time to simplify bookkeeping, despite the decrease in performance. Satisfying these two competing objectives, increasing efficiency through the largest possible step sizes while ensuring events are properly detected, is the central challenge addressed here.

3. RELATED WORK

Excellent works on the event detection problem in general are cited above; however, it is only recently that specialized modelling languages [Alur et al. 2000] and simulation environments [Deshpande and Semenzato 1995] have become available for large scale multi-agent hybrid systems. Relevant work includes two tangentially related fields: Multirate numerical integration methods and Distributed Discrete Event System Simulation (DDESS).

Given a group of coupled, purely smooth differential equations (i.e., no events) which inherently evolve at different time scales, *Multirate* integration techniques attempt to efficiently integrate the system of ODE's by using different integration rates, or step sizes for each equation (see Gear and Wells [1984], Engstler and Lubich [1996], or Esposito and Kumar [2001], for example). By using the appropriate size time step for each ODE it is hoped that the overall computational effort for the system will be reduced. Significant challenge lies in attempting to simulate two coupled differential equations using different time meshes. Often one must evaluate the derivative of the so-called fast variables at a point in time at which values of the slow variables have not been computed. This is typically accomplished by constructing some appropriate interpolant for the slow variables so that they may be approximated at off mesh points, when needed.

In spirit, the goal of multirate simulation is the same as the goal of this work: reduced computation time through asynchronous integration. However, in practice the challenges are quite different. The primary concern of multirate methods is accommodating coupling between the right-hand sides of two ODEs being integrated at different rates–a nonexistent issue in the multiagent problem because all coupled ODE's in our scheme are simulated at the same rate.

Likewise multirate methods do not address the problem of locating discrete events—the central concern in this article.

A second more closely related area is *Distributed Discrete Event System Simulation* (DDESS). Given a set of loosely decoupled but interacting Discrete Event Systems (DES), the idea is to make the best use of any available computational resources by simulating each DES on a different computer on a local network. Each DES is referred to as a "process" and is simulated in isolation. When a DES generates a message which is relevant to another DES, a message is sent over the network to the computer that is simulating that process. Complications arise because at any given instant, some of the individual process simulations may have progressed further along than others. This motivates the introduction of "local clocks" since there is no true notion of global time in an asynchronous distributed simulation. When a process sends a message, it appends to the message a time stamp which bears the value of its local clock at the instant when the message was generated. The contents of this message may alter the evolution of the receiving process; however the time-stamp will generally not match the receiving process's local clock. This means that the message may be in the "past" or "future". Future events pose little problem, however handling past events requires that the process must undo or "roll back" its simulation to a point where its local time clock is earlier than the time at which the message was generated. This necessarily implies that the process must store a history of its evolution so that it may always roll the simulation back. Roll back becomes expensive when many processes are involved because the rollback of one process may in turn trigger the rollback of another and so on, resulting in what is called a cascading rollback.

An active area of research in DDES concerns finding the best way to minimize rollback. Approaches fall into two categories: conservative and optimistic. Conservative approaches only allow each process to proceed up until a certain safe time [Fujimoto 2000] (a time before which it can be guaranteed no rollback will be necessary). Some processes must wait idle upon reaching this point in their simulation until other processes catch up resulting in underutilized resources.

In Jefferson [1985], the Time Warp Algorithm was introduced which was the first *optimistic* simulation methodology. In the optimistic approach each process is allowed to advance at its own rate hence no resources sit idle; however frequent rollback is possible. Jefferson's algorithm included an elegant mechanism called antimessaging to reliably deal with rollback. Since then there has been an explosion of research in the field exploring the tradeoff between idle resources and the cost of rollback (see, e.g., Lubachevsky [1989], Steinman [1992], or Carothers et al. [1999]). Many are still built on the Timewarp algorithm. They explore such issues as the best strategy for periodically exchanging information to minimize rollback. Many also consider topics specific to distributed computation such as network delays and bandwidth. It is obvious that this work bears relation to multi-agent hybrid systems; however since the systems are purely discrete there are no challenges related to integrating the differential equations. Also, the challenges associated with implementing a simulation algorithm on multiple processors are beyond the scope of this article.

Mirtich [2000] introduced a multibody Newtonian mechanics simulator for graphics applications which is built upon the Time Warp algorithm [Jefferson 1985]. In this framework, each rigid body is considered a process with its own local clock; collisions and other changes in contact states (rolling, sliding, etc.) play the role of messages or discrete events. This framework permits the use of an asynchronous integration procedure. Although there is some additional overhead associated with the method as the number of bodies increases, the performance of the method appears to be superior to traditional synchronous rigid body simulators by approximately one order of magnitude. It was shown that these savings increase as the number of agents grow. The goal of this work is identical in spirit to Mirtich's—reduce the computational effort required to simulate large multi-agent systems through asynchronous simulation and careful handling of discrete events. The key difference is that Mirtich's work exploits many unique features of Newtonian Mechanics and uses collision detection algorithms, specific to polyhedral bodies, to detect "events". This specificity allows the simulator to achieve only slightly subrealtime performance; however, it is not general enough to apply to arbitrary hybrid systems. Similar issues are addressed in Nicol and Perrone [2000] which addresses the special structure of wireless communication systems.

A second important related work is Hur and Lee [2002], which introduced an algorithm for distributed (and asynchronous) multi-agent hybrid system simulation. Several heuristics for determining the frequency with which global state information should be broadcasted. The most obvious difference between this work and ours is that the algorithms and implementation are distributed in Hur and Lee [2002], and therefore different issues must be addressed. A more subtle, yet conceptually important, distinction is that the algorithms presented in Hur and Lee [2002] represent *optimistic* approaches to the simulation problem—meaning that they accept possibly frequent rollback in the local time clocks. Our work presented here can be viewed as, to borrow from the DDES literature, a *conservative* approach to the problem. We feel that, unlike DDES, the only way to truly guarantee that no events are missed in hybrid simulation is to only permit the local clocks to advance independently to the extent that it can be shown that no events will be missed. In many ways our work and Hur and Lee [2002] can be viewed as complementary approaches to multi-agent simulation.

4. APPROACH

In this section, we describe the idea behind our approach to selecting step sizes for the asynchronous integration in such a way as to reliably detect events, with minimal, if any, rollback. We defer details of the implementation, as well as other considerations for selecting step sizes until Section 5.

Given the Multi-Agent Hybrid System

$$\dot{x}^i = f^i(x^i), \quad i = 1, \ldots, N \tag{6}$$

$$g(x^1, \ldots, x^N) < 0, \tag{7}$$

where superscripts index the N agents. Let $x^i \in R^{n_i}$ be the state of Agent i, $f^i : R^{n_i} \to R^{n_i}$ the time derivative of state x^i, and $g : R^{n_1} \times \cdots \times R^{n_N} \to R$ is

the event function. Assume that f^i is continuous as long as $g(x^1, \ldots, x^N) < 0$ and that g itself is smooth. Note that with no loss of generality, one may assume that $g(x^1, \ldots, x^N)$ is linear in the states since systems with a nonlinear event function can be converted to an equivalent system with a linear event function by appending an extra state variable as follows [Gear and Osterby 1984]:

$$\dot{x}^i = f^i(x^i)\, i = 1, \ldots, N \tag{8}$$

$$\dot{x}^{N+1} = \frac{\partial g}{\partial X} \cdot [f^1 f^2 \ldots f^N]^T \tag{9}$$

$$\tilde{g} = x^{N+1} < 0, \tag{10}$$

where X is the global state vector. For now, we assume all event functions are linear.

Since differential equations are numerically integrated using difference equations and the states are approximated at a discrete set of points in time $T \in \{t_1, t_2, \ldots, t_k, \ldots\}$, let subscripts denote the step, or iteration, number. Asynchronous simulation implies there is no single global value of time, t_k, instead the simulation of each agent proceeds independently, whenever possible. As such, local times and time steps, must be defined for each agent and they are also denoted using superscripts. Therefore, any quantity with a superscript i and subscript k, refers to agent i and has been evaluated at t_k^i. For example, $x_k^i = x^i(t_k^i)$ and $f_k^i = f^i(x^i(t_k^i))$. Also define the integration time step $h_k^i = t_k^i - t_{k-1}^i$.

As in Esposito et al. [2001b], we choose to use Predictor—Corrector numerical methods to integrate the system (see any standard monograph on numerical integration such as Ascher and Petzold [1998]). Applying the Predictor difference equation to the state

$$x_{k+1}^i = x_k^i + h_{k+1}^i \sum_{j=1}^{m} \beta_j\left(h_{k+1}^i\right) f_{k-j+1}^i, \tag{11}$$

where the β's are weighting functions which are polynomial functions of the step size.

When Eq. (11) is substituted into $g(x)$ the change in the value of the event function with each iteration is given by,

$$g_{k+1} = g\left(x_k^1 + h_{k+1}^1 \sum_{j=1}^{m} \beta_j\left(h_{k+1}^1\right) f_{k-j+1}^1, \ldots \right.$$
$$\left. x_k^N + h_{k+1}^N \sum_{j=1}^{m} \beta_j\left(h_{k+1}^N\right) f_{k-j+1}^N \right) \tag{12}$$

or, by linearity,

$$g_{k+1} = g_k + h_{k+1}^1 \frac{\partial g}{\partial x^1} \sum_{j=1}^{m} \beta_j\left(h_{k+1}^1\right) f_{k-j+1}^1$$
$$+ h_{k+1}^2 \frac{\partial g}{\partial x^2} \sum_{j=1}^{m} \beta_j\left(h_{k+1}^2\right) f_{k-j+1}^2 + \cdots$$

$$+ h_{k+1}^N \frac{\partial g}{\partial x^N} \sum_{j=1}^{m} \beta_j\left(h_{k+1}^N\right) f_{k-j+1}^N. \tag{13}$$

Using a control system analogy, Eq. (13) is analogous to a system with a single output, g, and N inputs, $h_{k+1}^1, \ldots, h_{k+1}^N$, Esposito et al. [2001a]. Now our goal is to select these N inputs in such a way as to properly "steer" the system to any zeros of g. However, it is important to point out that simply driving $g(x^1(t^1), \ldots, x^N(t^N)) \to 0$ is *not* sufficient to detect an event. In fact $g = 0$ does not correspond to a physical event *unless* $t_{k+1}^1 = t_{k+1}^2 = \cdots = t_{k+1}^N$. If $g = 0$, but $t_{k+1}^l \neq t_{k+1}^j$, this implies that the two agents have passed though the same point on the event surface *but at different local times*. Such a situation is not a state event, it is simply an artifact of the asynchronous simulation. Returning to the automated vehicle example, if the guard signified collisions between the vehicles (i.e., $g(x)$ is separation distance), the case in which $g = 0$ but $t_{k+1}^1 \neq t_{k+1}^2$ would imply the cars passed through the same point in the configuration space at different times, which obviously does not constitute a collision.

Fortunately, the fact that there are more inputs than outputs implies that there is some freedom in the selection of the step sizes. This latitude may be used to accommodate secondary criteria. Thus $N-1$ additional inequalities are added to the original event criteria, which we call *synchronization constraints*. Since the labelling of the agents is arbitrary, assume that $t_{k+1}^1 \geq t_{k+1}^2 \geq \cdots \geq t_{k+1}^N$. The $N-1$ synchronization functions would be defined as

$$\tau_{k+1}^i = t_{k+1}^i - t_{k+1}^1 < 0, \quad i = 2, \ldots, N, \tag{14}$$

each measuring the extent to which an individual agent's time clock lags the leading agent's clock.

Physically meaningful events are now defined as

$$(g = 0) \wedge \left(\tau_{k+1}^2 = 0\right) \wedge \cdots \wedge \left(\tau_{k+1}^N = 0\right). \tag{15}$$

One must then select $h_{k+1}^1, \ldots, h_{k+1}^N$ so that the simulation will precisely land on the point in time when all of these conditions are true. Collectively, the dynamics of the event and synchronization functions can be written as follows.

$$\begin{bmatrix} g \\ \tau^2 \\ \vdots \\ \tau^N \end{bmatrix}_{k+1} = \begin{bmatrix} g \\ \tau^2 \\ \vdots \\ \tau^N \end{bmatrix}_k + \tag{16}$$

$$\begin{bmatrix} \frac{\partial g}{\partial x^1} \sum f^1 \beta(h^1) & \frac{\partial g}{\partial x^2} \sum f^2 \beta(h^2) & \ldots & \frac{\partial g}{\partial x^N} \sum f^N \beta(h^N) \\ -1 & 1 & 0 \ldots & 0 \\ -1 & 0 & 1 \ldots & 0 \\ & \vdots & & \\ -1 & 0 & 0 \ldots & 1 \end{bmatrix}_k \begin{bmatrix} h^1 \\ h^2 \\ \vdots \\ h^N \end{bmatrix}_{k+1}.$$

Inspired by the feedback control analogy our step size selection scheme is essentially a discrete time version of the *feedback linearization* technique from nonlinear control theory, which uses a judicious selection of the input to cancel the system's nonlinear dynamics, rendering it a stable linear system.

THEOREM 4.1. *By selecting choosing a value of the "gain"* $0 \leq \gamma < 1$*, defining*

$$h_{k+1}^i = (\gamma - 1)\tau_k^i + h_{k+1}^1, \quad i = 2, \ldots, N \tag{17}$$

and selecting h_{k+1}^1 *as the smallest positive real root of the following polynomial*

$$\begin{aligned}(1-\gamma)g_k \; + \; & h_{k+1}^1 \frac{\partial g}{\partial x^1} \sum_{j=1}^{m} \beta_j\big(h_{k+1}^1\big) f_{k-j+1}^1 \\ & + \; \big((\gamma-1)\tau_k^2 + h_{k+1}^1\big)\frac{\partial g}{\partial x^2} \sum_{j=1}^{m} \beta_j\big((\gamma-1)\tau_k^2 + h_{k+1}^1\big) f_{k-j+1}^2 \\ & + \cdots + \big((\gamma-1)\tau_k^N + h_{k+1}^1\big)\frac{\partial g}{\partial x^N} \sum_{j=1}^{m} \beta_j\big((\gamma-1)\tau_k^N + h_{k+1}^1\big) f_{k-j+1}^N\end{aligned} \tag{18}$$

all events will be properly detected (*within the tolerance level of the integration*) *without having to roll the simulation back* (*i.e., decrement* k).

PROOF. Substituting Eq. (17) into Eq. (16) produces the following difference equations for the synchronization function

$$g_{k+1} \; = \; g_k + h_{k+1}^1 \frac{\partial g}{\partial x^1} \sum_{j=1}^{m} \beta_j\big(h_{k+1}^1\big) f_{k-j+1}^1 \tag{19}$$

$$\begin{aligned} & + \big((\gamma-1)\tau_k^2 + h_{k+1}^1\big)\frac{\partial g}{\partial x^2} \sum_{j=1}^{m} \beta_j\big((\gamma-1)\tau_k^2 + h_{k+1}^1\big) f_{k-j+1}^2 + \ldots \\ & + \big((\gamma-1)\tau_k^N + h_{k+1}^1\big)\frac{\partial g}{\partial x^N} \sum_{j=1}^{m} \beta_j\big((\gamma-1)\tau_k^N + h_{k+1}^1\big) f_{k-j+1}^N \end{aligned} \tag{20}$$

$$\tau_{k+1}^i \; = \; \gamma\tau_k^i, \quad i = 2, \ldots, N. \tag{21}$$

Further substituting the roots of Polynomial (18) for h_{k+1}^1 will yield

$$g_{k+1} \; = \; \gamma g_k \tag{22}$$

$$\tau_{k+1}^i \; = \; \gamma\tau_k^i. \tag{23}$$

The solution to the above difference equations is

$$g_k \; = \; (\gamma)^k g_0 \tag{24}$$

$$\tau_k^i \; = \; (\gamma)^k \tau_0^i. \tag{25}$$

Provided $0 \leq \gamma < 1$, as $k \to \infty$, $g_k \to 0$ and $\tau_k^i \to 0$ for all $i = 2, \ldots, N$, implying that the simulation will terminate at the event. Furthermore, if initially $g, \tau^1, \ldots, \tau^N < 0$, this equilibrium point is approached strictly from the right. This implies that there is no danger of advancing the local clocks too far, eliminating the need for rollback.

Several observations are worth pointing out.

(1) The step size selection suggested above represents the ideal step size based on event considerations. In practice, the step size will occasionally be limited further to reduce truncation error in the integration.
(2) Polynomial (18) is essentially the most accurate extrapolation polynomial that can be constructed which is consistent with the underlying integration accuracy. Since underlying state estimates are generated by a forward integration scheme it does not make senses to use the negative real root(s) of this polynomial. Obviously, complex roots have no physical meaning as time steps.
(3) If there are no positive real roots of the polynomial, then from the point of view of event detection, the acceptable step size is unbounded.
(4) If $\gamma = 0$, the system will converge in a single iteration.
(5) If the event function is nonlinear the technique is still applicable, however the statement regarding the need for rollback must be relaxed. The above choice of step size will still cancel the first and most dominant terms of the Taylor Series expansion of Eq. (12), however the uncancelled higher-order terms may act as forcing functions and introduce the possibility of overshooting the exact event time by at most one step. This is discussed further in Esposito [2002].

5. SIMULATION ALGORITHM

The multi-agent simulation algorithm, shown below, is referred to as *MAsim*. Figure 1 illustrates the simulation algorithm graphically; while the pseudo-code in Algorithms 1 and 2 provide more detail. The process illustrated in Figure 1 resembles a traditional hybrid system simulator, using a Predictor–Corrector integration algorithm, in many ways. First an initialization phase is required because Predictor–Corrector integrators require a history of derivative values. Different implementations achieve this start up process in different ways. Note that the user specifies ϵ, the integration error tolerance which should not be exceeded. The step size computation, as discussed in Section 4 is done by the routine Algorithm 2, called *MAstepSelect*.

An important issue in asynchronous simulation is deciding the order in which the individual agents should be advanced. In this case, at each iteration in the simulation the agent whose local time clock lags the most is always selected to be integrated. In the case of a tie, an agent may be chosen at random among the agents with the smallest local times. At the end of each iteration, the agents are relabelled so that $t^1_{k+1} \geq t^2_{k+1} \geq \cdots \geq t^N_{k+1}$ to ensure that $\tau^i_{k+1} \leq 0$. Therefore, at each iteration, agent-N is always selected for advancement. This process of consistently selecting the agent with the lagging time clock has the effect of bounding the maximum amount of time by which the local time clocks may disagree. In effect,

$$\max_{i=2,\ldots,N} [\tau^i] \leq h_{\max}, \tag{26}$$

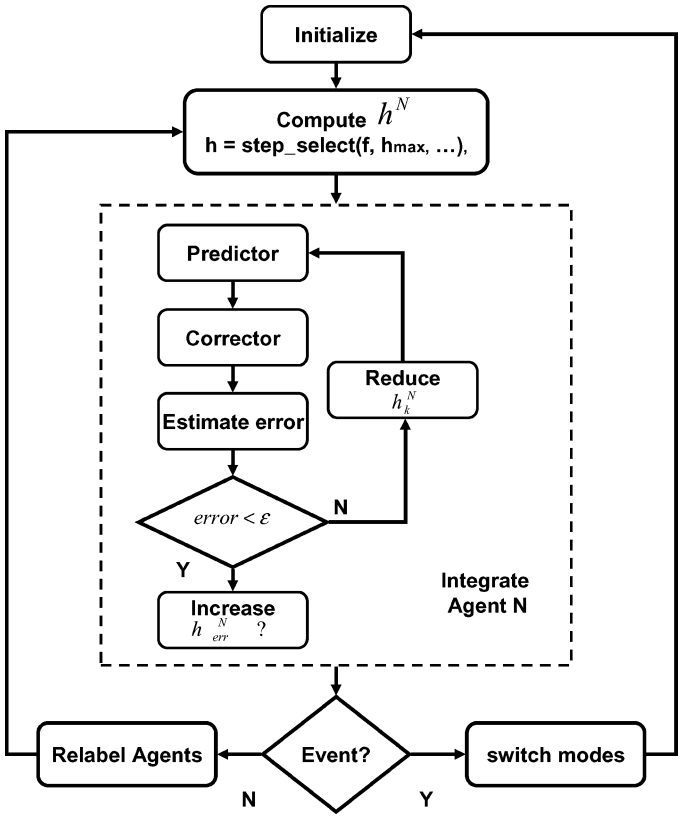

Fig. 1. The overall simulation algorithm *MAsim* presented in Section 5.

where $h_{\max}$ is the prespecified maximum allowable step size. Since the event detection scheme is essentially based on solving for the roots of an extrapolation polynomial at each step, bounding the extrapolation interval helps to ensure meaningful estimates of the event time.

In Figure 1, everything inside of the dashed box is accomplished using the standard integration technique along with the Predictor–Corrector method. After each new state is constructed, the approximation error in the result is estimated and compared to a threshold. This information is used to decide if the result should be accepted or rejected as well as to determine what h_{err} should be to keep the integration error within the acceptable range for the next iteration.

Finally, it must be determined if an event has occurred. This determination must be made using our new requirement for event occurrence, namely $g_k = 0 \wedge \tau_k^2 = 0 \wedge \cdots \wedge \tau_k^N = 0$. If indeed one has occurred, the simulation is stopped, all mode switches are enabled and the simulation restarts in the new mode. Note that the initialization process must be repeated since the definitions of the derivatives have changed. If instead, it turns out that no event occurred, the agents are relabelled so that $t_k^1 \geq t^2 \geq \cdots \geq t_k^N$ to ensure that $\tau_k^i \leq 0$, as discussed above.

Algorithm 1 *MAsim*: Multi-agent simulation algorithm

0. Given initial conditions $x^1(0), \ldots x^N(0)$; user defined values for $h_{\min}$, $h_{\max}$, and ϵ; and initial estimates for $h^1_{err} \ldots h^N_{err}$.
1. Initialize the integration algorithm; let $k = 1$.
2. Compute $h^N_k = \text{MAstepSelect}(f^1_k, \ldots, f^1_{k-m}, \ldots, f^N_k, \ldots, f^N_{k-m}, h_{\max}, h_{\min}, h^N_{err}, \frac{\partial g}{\partial x})$.
3. Integrate Agent N using h^N_k via the Predictor Corrector Algorithm. Estimate error e^N.

if $e_N < \epsilon$ **then**
  Store result.
  Compute h^N_{err} for next iteration.
  $k \leftarrow k + 1$.
else
  Reduce h^N_k.
  Goto 3.
end if
if $g = 0 \wedge \tau^2 \wedge \cdots \wedge \tau^N$ **then**
  Switch modes.
  Goto 1.
else
  Relabel agents $1, \ldots, N$ in order of descending values of t^i.
  Goto 2.
end if

Algorithm 2 *MAstepSelect*: The step size selection subroutine for *MAsim*.

$h^N = \text{MAstepSelect}(f^1_k, \ldots, f^1_{k-m}, \ldots, f^N_k, \ldots, f^N_{k-m}, h_{\max}, h_{\min}, h^N_{err}, \nabla g)$ {
for i = 1:N **do**
  $\tau^i = t^i - t^1$.
end for
Z = Roots of Polynomial (18).
$R = \{r | r \in Z, Im(r) = 0, Re(r) \geq 0\}$.
if $R = \emptyset$ **then**
  $h^N_g = \infty$.
else
  $h^1_g = \min(R)$.
  $h^N_g = (\gamma - 1)\tau^N_k + h^1_g$.
end if
$h^N = \max[h_{\min}, \min(h_{\max}, \min[h^N_g, h^N_{err}])]$. }

Turning our attention to the subroutine that selects the integration step sizes, there are several issues warranting explanation. Note that even though routine seeks to compute h^N, h^1 must be computed first because h^N is defined in terms of h^N as in Eq. (17). Step sizes with a subscript *err* denote candidate step sizes computed on the basis of controlling the truncation error; while the subscript g denotes a step size computed based on event detection criteria. As discussed in Section 4, the ideal step size from the point of view of event detection is computed according the smallest real positive roots of the polynomial. If there are no such root, then h_g is set to infinity. This implies that, from an event

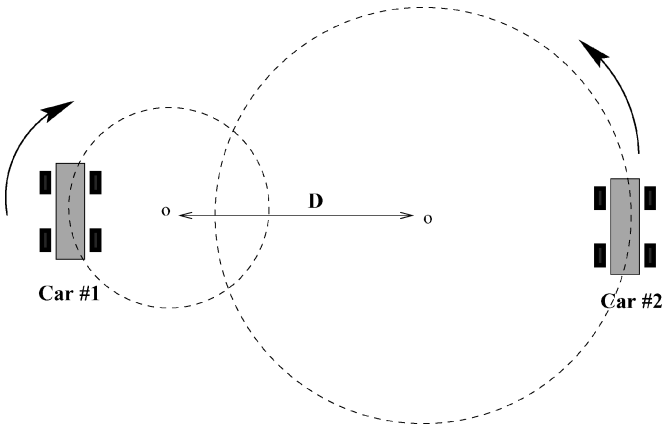

Fig. 2. Two automated vehicles driving in spiral trajectories.

detection perspective, there is no constraint on the step size. Finally, note that, as with any simulation, event proximity is not the only consideration when choosing a step size. Typically, there are user defined minimum and maximum step sizes, $h_{\min}$ and $h_{\max}$, as well as an estimate of the ideal step size based on the truncation error h_{err}. The correct choice is

$$h_{k+1} = \max[h_{\min}, \min(h_{\max}, \min[h_{err}, h_g])]. \tag{27}$$

6. EXAMPLES

In this section, the effectiveness of the algorithm is demonstrated on the two car example used to motivate the problem in Section 1, also illustrated in Figure 2. Events occur when the two autonomous vehicles come within R feet of each other, in practice this condition would signal an imminent collision and a mode switch would activate an emergency collision avoidance controller.

The dynamics for agent $i = 1, 2$ are given in Eq. (2) and Eq. (3); while the guard is given in Eq. (5). The v^i and ω^i are state feedback laws (i.e., functions of x^i, y^i and θ^i) and are selected in such a way as to steer the vehicles in spiral trajectories. However, Car 1 travels much faster than Car 2. This example was simulated using the algorithm presented earlier for two scenarios–each with a different value for the of initial states and separation distance.

In the first example (see Figure 3 and 4) the value of D is small enough that the robots eventually collide. The path of the robots in the plane is shown in Figure 3 (left). Figure 3 (right) is a plot of the value of the event function versus the step number k. The value peaks first around step 70, when Robot 1 completes a half of a rotation; finally at step number 107 the event detection criterion becomes active and the step sizes are selected in such a way that $g \rightarrow 0$ exponentially. Further insight is gained by examining Figure 4 (left), which shows the step sizes used for each agent as a function of time. Note how h^1 and h^2 vary independently until $t \approx 9.8$ when the

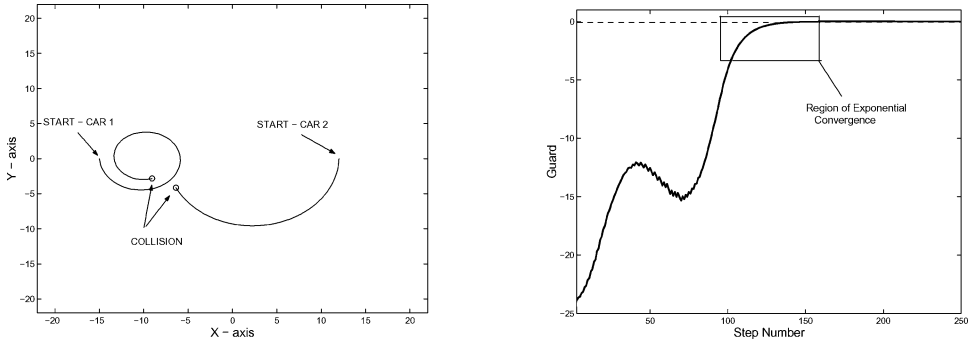

Fig. 3. The trajectory of the two robot vehicles in the plane is plotted (left); it is of interest to detect when they come within R feet of each other. The value of the event function (separation distance) as a function of step number is shown (right). The event function value converges to zero exponentially.

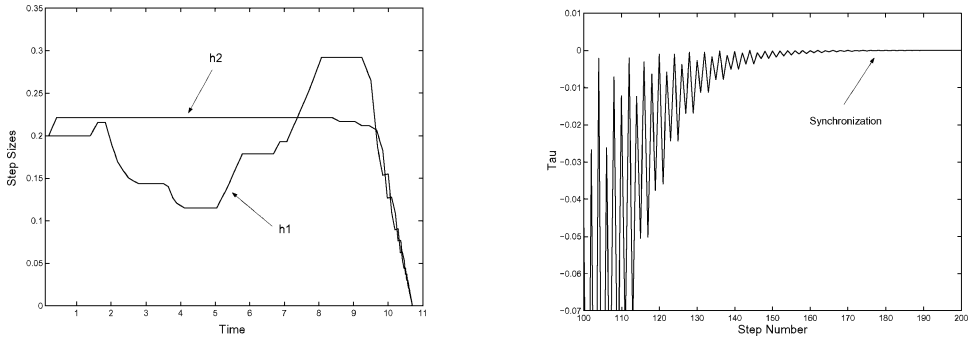

Fig. 4. Step sizes used in the first example are shown (left). h^1 and h^2 are selected independently away from the constraint but are brought into synchronization when an event is impending. The value of τ (right), which is a measure of the discrepancy between the two local time clocks (i.e., asynchrony), rapidly decreases in magnitude as the event is approached.

event detection criteria begins constraining the step sizes; whereafter they both begin adjusting the step sizes so as to synchronize the two local clocks. This is further illustrated in Figure 4 (right), which plots the history of the synchronization function τ. Its value rapidly approaches zero, at which point $t^1 = t^2$.

In the second case (see Figures 5 and 6), the value of the initial separation distance is large enough that the vehicles do not collide, but they do come very close to one another. The trajectories of the two vehicles are shown in Figure 5 (left). There is a near miss halfway through Agent-1's second rotation. The history of the constraint in Figure 5 (right) shows that the value of $g(x)$ comes very close to zero around step number 150. During this period, the value of τ, Figure 6 (left), decreases in preparation for a possible event. Figure 6 (right) shows how the step sizes were decreased to slow down the simulation. However once the two vehicles pass each other and it becomes apparent that no collision will occur, the step sizes quickly increase and the two local clocks are allowed to fall out of synchronization again. Note that in these simulations, $\gamma = 0.5$ so

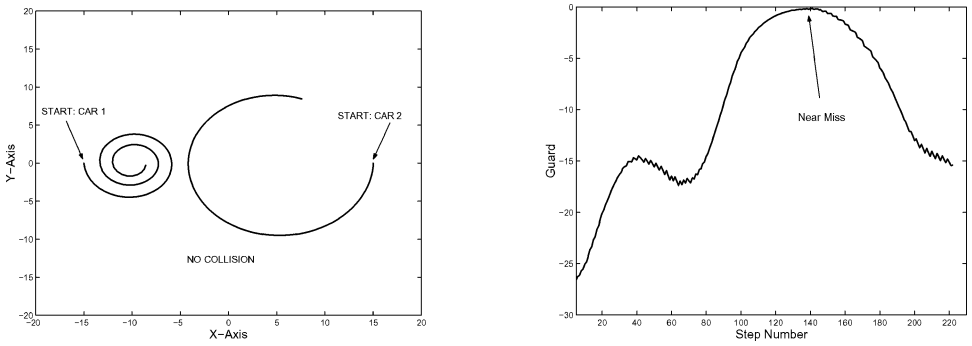

Fig. 5. The trajectory of the two robot vehicles in the plane for the second example (left). The initial condition is selected such that the two vehicles do not come within R feet of one another. However they do approach quite closely. The value of the event function as function of the step number is shown (right).

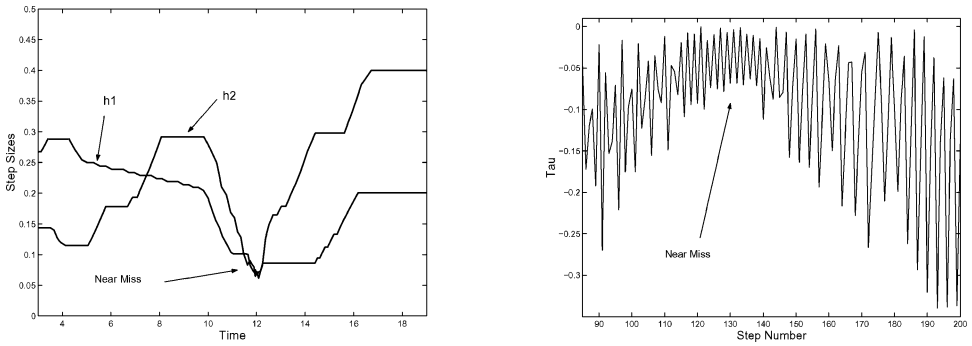

Fig. 6. The step sizes used in the second example are shown (left). Again, h^1 and h^2 are selected to synchronize the simulation when it seems an event may occur. Once it is apparent that this is not the case, the simulation returns to asynchrony. The magnitude of τ (right), therefore decreases during this period but soon increases again.

that the "slow down" and synchronization, as the state approached the guard surface, could be more easily illustrated. In practice a value closer to zero would produce rapid convergence.

7. EFFICIENCY ANALYSIS

Recall that the primary motivation for using asynchronous simulation is to improve the efficiency of the simulator. In this section, the predicted performance of the asynchronous algorithm is modelled statistically under some assumptions and compared to that of the more traditional asynchronous algorithm.

7.1 The Traditional Algorithm

For the sake of comparison, the traditional synchronized algorithm is reviewed here. The traditional algorithm makes no distinction between single agent systems and multi-agent system. In both cases it treats the entire state vector as if it belongs to a single system by using a single global step size; we refer

Algorithm 3 *SAsim*: The traditional (i.e., single agent) hybrid system simulation algorithm.

Given initial conditions $x^1(0), \ldots, x^N(0)$; user defined values for $h_{\min}, h_{\max}$, and ϵ; and initial estimates for h_{err}.
1. Initialize the integration algorithm. Let $k = 1$.
2. Compute $h_k = \text{SAstepSelect}(f_k^1, \ldots, f_{k-m}^1, \ldots, f_k^N, \ldots, f_{k-m}^N, h_{\max}, h_{\min}, h_{err}^N, \nabla g)$.
3. **for**i = 1:N **do**
   Integrate x^i using h_k.
   Estimate error e^i.
end for
$e = \max(e^1, \ldots, e^N)$.
if $e < \epsilon$ **then**
   Store result.
   Compute h_{err} for next iteration.
   $k \leftarrow k + 1$.
else
   Reduce h_k.
   Goto 3.
end if
if $g = 0$
   Switch modes.
   Goto 1.
else
   Goto 2.
end if

Algorithm 4 *SAstepSelect*: The step size selection algorithm used by *SAsim*, the traditional simulation algorithm

$h = \text{SAstepSelect}(f_k^1, \ldots, f_{k-m}^1, \ldots, f_k^N, \ldots, f_{k-m}^N, h_{\max}, h_{\min}, h_{err}, \nabla g)$ {
$Z = \text{Roots}\ (P(h))$.
$R = \{r | r \in Z, Im(r) = 0, Re(r) \geq 0\}$.
if $R = \emptyset$ **then**
   $h_g = \infty$.
else
   $h_g = \min(r)$.
end if
Select final step size $h = \max(h_{\min}, \min(h_g, h_{err}, h_{\max}))$.

to this as the *Single Agent Algorithm* or *SAsim*. The notation follows the same convention as in the Multi-agent Algorithm *MAsim*.

While any suitable integration method or extrapolation polynomial can be used, since the goal of this discussion is to illuminate the pros and cons of asynchronous simulation with respect to the approach of using a single global step size we assume the numerical integration in both cases is performed using the same algorithm (Predictor–Corrector) and a very similar approach to event detections such as that of Esposito et al. [2001b]. The polynomial $P(h)$ which appears in the `step-select` routine within the *SAsim* algorithm is essentially

a "single agent" version of Eq. (18)

$$(1-\gamma)g_k \; + \; h_{k+1}\left[\frac{\partial g}{\partial x^1}\sum_{j=1}^{m}\beta_j(h_{k+1})f^1_{k-j+1} + \cdots + \frac{\partial g}{\partial x^N}\sum_{j=1}^{m}\beta_j(h_{k+1})f^N_{k-j+1}\right]. \tag{28}$$

Note that the superscript on h has been dropped because there is a single global step size; however the polynomial has the same effect of feedback linearizing the guard dynamics.

There are two primary differences between the *SAsim* and *MAsim* algorithms. In *SAsim*, a single ideal step size from the point of view of event detection h_g is computed using, while in *MAsim* the event dynamics in Eqs. (18) and (17) are used to compute N distinct step sizes, one for each agent. Obviously, doing this requires significantly more overhead in computing the various τ's, h's and relabelling the agents. The underlying polynomial used in *MAsim* is also slightly more complex.

The second difference is that in *SAsim* a single step size from the point of view of the truncation error is computed. This step size is based on the *maximum* of all the agents truncation error. This results in using the *minimum* step size across all agents. Obviously, this implies the *SAsim* algorithm will, in general, use smaller time steps and therefore requires more steps to complete the simulation.

7.2 Average Complexity Analysis

Overall *MAsim* requires more computation per iteration; while *SAsim* requires more iterations to complete a simulation. In this section, we attempt to quantify this tradeoff by studying their affect on the quantity, which we call the Speed-up Factor, defined as

$$S = \frac{C_{MA}}{C_{SA}}, \tag{29}$$

where C_{MA} is the computational cost (floating point operations or CPU time) required to simulate a specific system using *MAsim* and C_{SA} is the cost of simulating the same scenario using *SAsim*, assuming all parameters are equal, such as the integration tolerance. Values of S significantly smaller than one imply that a great speedup is obtained using *MAsim*.

Obviously, S is entirely problem (and implementation) dependent therefore we choose to instead study the *average* Speed-up Factor. The assumption is that since many complex factors, which cannot all be modelled, influence the step size selection criteria we choose to make the reasonable modelling assumption that the step size that the algorithm selects is a uniformly distributed random variable $h^i \in [h_{\min}, h_{\max}]$. This assumption enables us to focus on the general effects of the tradeoff between having a higher overhead in the case of *MAsim* versus being forced to use smaller step sizes in the case of *SAsim*. Note that the analysis can be repeated using any distribution on the step sizes; however, the uniform distribution is chosen because it does not clearly favor either algorithm. In that sense, it is arguably the most "neutral" distribution. Further discussion is postponed to Section 8.

Since $C_{SA} = E[h_{SA}]CI$ and $C_{MA} = E[h_{MA}](CI + OH)$ where $E[h_{SA}]$ and $E[h_{MA}]$ are the expected values of the step sizes chosen by the two algorithms; OH is the *additional* overhead associated with the *MAsim* algorithm; and C_I is the cost of integrating a single agent through a single time step. Note that C_I is identical in both algorithms, provided the same integration technique is used. Therefore,

$$S_{avg} = \frac{E[h_{SA}]}{E[h_{MA}]} \cdot \left\{1 + \frac{OH}{C_I}\right\}. \tag{30}$$

S_{avg} then depends on two factors: the ratio of the expected values of the step size selected by each algorithm $E[h_S A]/E[h_{MA}]$; and the size of **MA**'s overhead in comparison with the cost of integrating though a single step, OH/C_I, (referred to as the *relative overhead*). This quantifies the discussion in Section 7.1.

7.2.1 *Expected Values of Step Sizes.* The assumption of a uniform distribution means that the expected value of the step size for the ith agent is simply

$$E[h^i] = \frac{(h_{\max} - h_{\min})}{2 + h_{\min}}. \tag{31}$$

In the case of the Multi-agent Algorithm, at each iteration step sizes $h^1, \ldots, h^N$ are computed and each agent is integrated with its respective step size. So $E[h_{MA}] = E[h^i]$ since the expected value does not depend on i. The distribution for h_{MA} remains the same *regardless of the number of agents*.

In the case of the Single-agent Algorithm $h^1, \ldots, h^N$ are computed in the same way; *however*, at each iteration, a single global time step must be selected for all agents due to synchronization requirements, so

$$h_{SA} = \min[h^1, \ldots, h^N]. \tag{32}$$

Therefore, one would expect the distribution of h_{SA} to be biased toward small step sizes (see Figure 7). It is difficult to write a closed for expression for $E[h_{SA}]$, however, a normalized histogram generated numerically appears in Figure 8. As N increases, one sees that the frequency with which small step sizes are selected increases dramatically in the Single-agent Algorithm. It can be shown that the expected value of this distribution takes the form

$$E[h_{SA}] \approx h_{\min} + C_1 \exp(-C_2 N), \tag{33}$$

where C_1 and C_2 are positive constants. As a result

$$\frac{E[h_{SA}]}{E[h_{MA}]} \approx \frac{h_{\min} + C_1 \exp(-C_2 N)}{(h_{\max} + h_{\min})/2}. \tag{34}$$

Therefore,

$$\lim_{N \to \infty} \frac{E[h_{SA}]}{E[h_{MA}]} = \frac{2h_{\min}}{h_{\max} + h_{\min}}. \tag{35}$$

7.2.2 *Computing the Relative Overhead.* The second factor influencing S_{avg} is the relative overhead, which is essentially the cost of completing any extra steps in *MAsim* not present in *SAsim* such as computing the synchronization events, relabeling the agents, and calculating extra step sizes. The

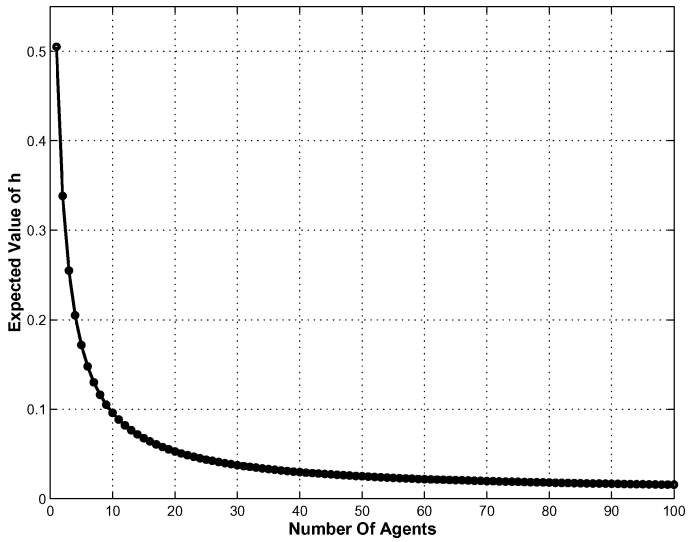

Fig. 7. The expected value of h (normalized), for the Single Agent Algorithm as a function of the number of agents.

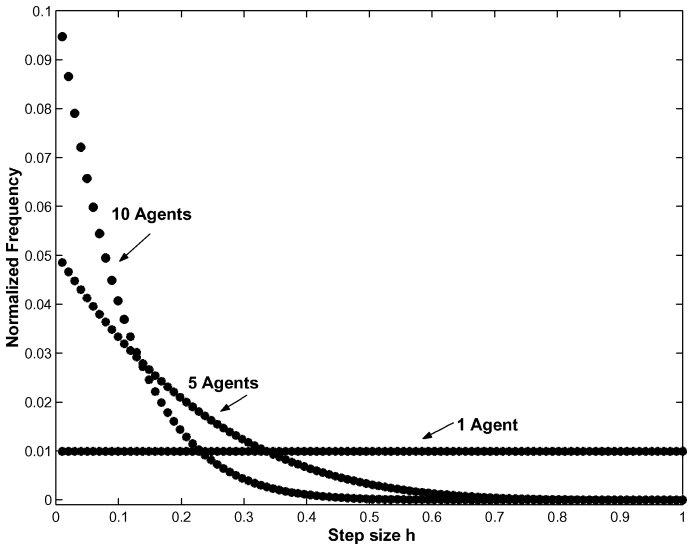

Fig. 8. A histogram depicting the frequency of selected step sizes (normalized) for the Single-agent algorithm, for several values of N (the number of agents), as the number of agents becomes large.

complexity of each these components of the overhead is $O(N)$ in the number of agents,

$$\frac{OH}{CI} \approx C_3 + C_4 N, \tag{36}$$

where C_3 and C_4 are positive constants. This was confirmed experimentally.

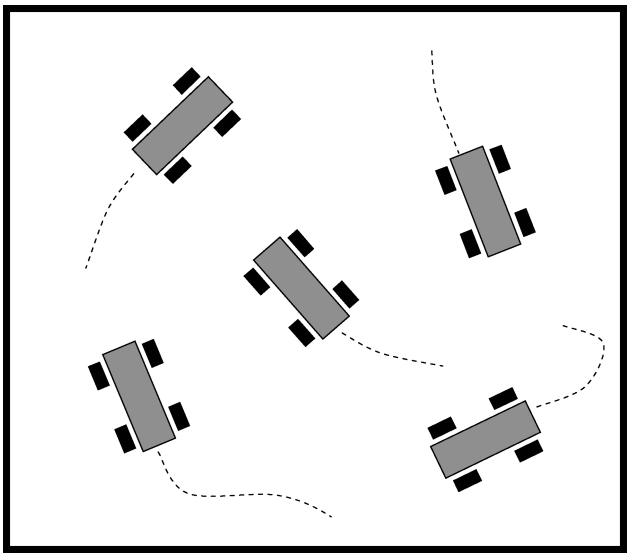

Fig. 9. The N-agent "bumper car" example. Events occur when an agent collides with one of the walls or another agent.

Recall that the relative overhead is the overhead as a percentage of the cost of integrating through a single step C_I. Naturally C_I is highly problem dependent but it is at *least* 10 floating point operations. Problems with more complicated expressions for $f(x)$ typically have higher values of C_I. Such problems favor *MAsim* even further.

7.2.3 *Prediction and Discussion.* Using (30), (34), and (36) on can then predict the behavior of S_{avg} as a function of the number of agents,

$$S_{avg} \approx \frac{(h_{\min} + C_1 \exp^{-C_2 N})(C_3 + C_4 N)}{(h_{\max} + h_{\min})/2}, \tag{37}$$

where the particular values of the constants are problem dependent.

8. NUMERICAL EXPERIMENTS

The benchmark problem chosen to test these ideas is a larger version of the autonomous vehicle test problem first discussed in Section 1 and presented in Section 6. N kinematic car-like agents are confined to a rectangular "arena". Anytime an agent comes within a certain distance from a "wall" or another vehicle, an event occurs and an emergency control law is activated. See Figure 9. The dynamics are given by Eq. (2) and the functions v and ω were arbitrarily chosen. When a collision is about to occur with either another agent or with the boundary of the arena, the control law switches to avoid a collision. The "agent-wall" collision events are

$$g_l^i = c_{lw} - x^i \leq 0 \tag{38}$$

$$g_r^i = x^i - c_{rw} \leq 0 \tag{39}$$

$$g_t^i = y^i - c_{tw} \leq 0 \tag{40}$$

$$g_b^i = c_{bw} - y^i \leq 0 \tag{41}$$

where c_{lw} and c_{rw} and are the x coordinates of the left and right walls, and c_{tw} and c_{bw} and are the y coordinates of the top and bottom walls. The "agent–agent" collision event is

$$g^{ij} = (x^i - x^j)^2 + (y^i - y^j)^2 - R^2 \leq 0, \quad \forall j \neq i, \tag{42}$$

where R is the range at which the emergency control law kicks in. The number of differential equations is $3N$ and the number of guards is $4N + N \cdot (N-1)/2$.

The algorithms *SAsim* and *MAsim* were coded in Matlab so as to be as similar as possible to allow a fair comparison. Parameters such as the minimum and maximum step size, and integration tolerance were the same in both cases. The actual numerical integration algorithms were also identical.

Experiments were conducted to determine how the speedup factor S_{avg} varies with the number agents. Recall that it was predicted that as the number of agents increases, the Single Agent algorithm becomes increasingly likely to select smaller step sizes, while the Multi-agent Algorithm is not biased toward smaller step sizes. Data was collected for systems with 2 to 20 agents. For each number of agents, the system was simulated for 10 randomly generated initial conditions, using a relative error tolerence of 10^{-3}, for a fixed duration using both *SAsim* and *MAsim*. The number of floating point operations used by each algorithm for a given scenario was measured using the `flops` command in Matlab. The raw data is shown in Figure 10. Note for most of the 190 test scenarios, with the exception of 2 of the two agent scenarios, *MA* outperformed *SAsim*. In the case when $N = 20$, *SAsim* required, on average, double the amount of computations required by *MA*.

A second factor whose impact on performance was studied is the user-specified integration error tolerance. The performance improvement associated with *MA* would be expected to decline at tighter error tolerances. This is anticipated because tighter error tolerances have the effect of biasing the step size selection mechanism toward smaller step sizes. Each of the 190 scenarios mentioned above was simulated with three different integration errors tolerances, $\epsilon = 10^{-3}$, 10^{-4} and 10^{-5}, for a total of 570 experiments. The number of floating point operations used by each algorithm for a given scenario was measured.

Based on this data the speedup ratio for each scenario was calculated. S_{avg} for a given number of agents and a given tolerance was computed as the mean of the speedup ratios of the 10 random scenarios. The S_{avg} as a function of the number of agents is plotted in Figure 11. Each curve displays the results for a different value of the integration tolerance. Tighter integration tolerances have the effect of biasing the algorithms toward selecting smaller step sizes. The dashed lines in the figure shows the overall trend, qualitatively predicted by Eq. (37).

Clearly, the assumption that the selected step size is a uniformly distributed random variable is an idealization and cannot be used to make quantitative predictions on the performance. However, the data in Figure 11 and the shape

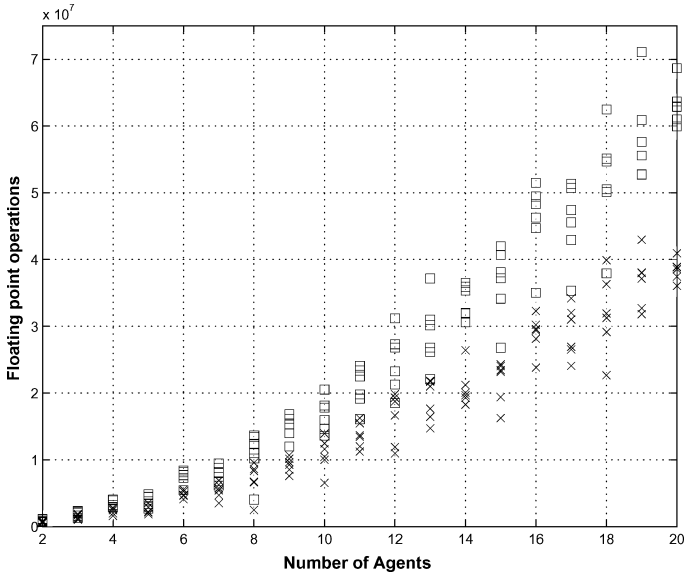

Fig. 10. Sample experimental results on the complexity of each algorithm as a function of the number of agents. Squares represent SAsim data and crosses are MAsim data.

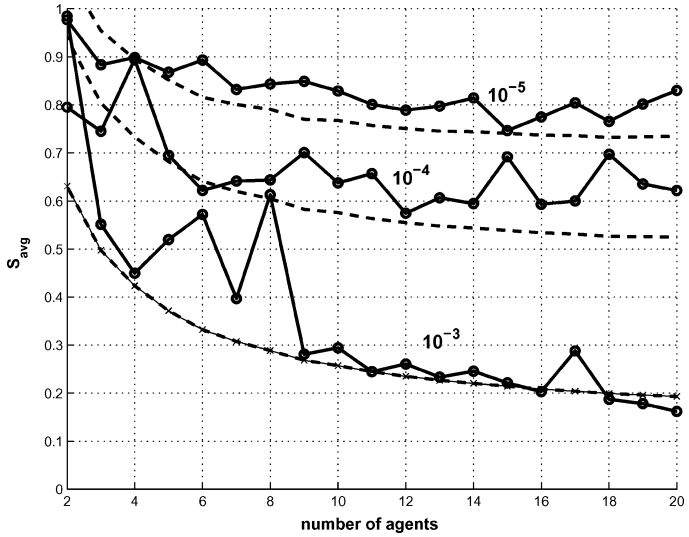

Fig. 11. Experimental results: Computational cost as a function of the number of agents, for various integration tolerances. Dash lines represent analytical predictions; while the solid line is the geometric mean of the experimental data.

of the dashed lines obtained by fitting Eq. (37) display markedly similar quantitative trends. The stochastic modelling exercise clearly captures the essence of the tradeoff at work between step size selection and overhead. It also indicates that a significant efficiency gain can be realized in practice (in the case of 20 agents a factor of 5 or more).

Of course, the actual step size "distribution" is nonuniform, which warrants a discussion how this assumption affects the analysis. Distributions which are non-uniform and heavily weighted around a mean will certainly bias the prediction depending on how the mean varies across all agents. If all the agents have the same distribution, with the same mean, this implies that the agents are all varying at nearly the same time scale, in which case little advantage is to be gained from an asynchronous simulation. On the other hand, if all agents have the same general distribution but their means *do not* coincide, the asynchronous simulation will be highly favored since this implies that some agents are consistently "slow" and others consistently "fast". In light of this, a uniform distribution was selected since it is does not *necessarily* favor either of these two extremes, representing a somewhat neutral assumption.

9. CONCLUSION

A major cause of performance loss when simulating large scale systems of ODEs can be traced back to the requirement that a single global step size is traditionally used despite the fact that the various individual differential equations in the system may each warrant a very different choice of step size. The only proper choice of a global time step among each individual equation's candidate time step is, unfortunately, the minimum of these candidates. As a result, the simulation often must proceed unnecessarily slowly using the smallest possible step size.

In situations in which the coupling between the right-hand sides of the ODEs is strong this cannot be avoided. However, we consider the special but important case of decoupled ODE's where the definition of the derivatives can change, or switch, when some function of the state variables changes sign. The switching conditions may introduce coupling between the subsystems. We refer to such systems as multi-agent hybrid systems. Examples include groups of automated highway vehicles, unmanned aerial vehicles, mobile robots, etc.—systems that are nominally physically decoupled, however, through sensing, software or actual physical collisions, interactions may arise which precipitate mode changes within the system.

In this article, a technique is introduced for simulating multi-agent hybrid systems in an asynchronous fashion, that is without resorting to a single global steps size. In this asynchronous simulation algorithm, each agent has its own local time clock. During parts of the execution in which no switches occur, the individual time clocks are allowed to advance at different rates as dictated by the agent's dynamics, allowing the maximum utilization of the available computational resources. This complicates the event detection problem because it is no longer sufficient to simply check the sign of the guard function since the individual states are being reported at different points in time. We introduced and algorithm that selects N integration step sizes for the N agents in such a way that the individual time clocks will synchronize on-the-fly when some relevant subset of the states approach a guard surface. The problem is analogous to a multi-input multi-output control system, for which the inputs are the N step sizes and the output functions are the guard function, along with $N - 1$

synchronization functions, which measure discrepancies between the local time clocks.

We show that while a single iteration of the new algorithm is somewhat more expensive due to the overhead associated with adaptive synchronization, not using a global time step requires fewer total iterations. Because the performance is somewhat problem dependent, we construct a probabilistic model of the step size selection scheme, assuming the step size for each agent is a random variable. Under the assumption that the distribution is uniform, it is shown that average complexity of the multi-agent algorithm is approximately quadratic in the number of agents being simulated. In contrast, it is demonstrated through experiments that, as the number of agents increases, the average computation time for the traditional algorithm seems to exponentially approach an upper bound.

Experiments were conducted in which both the traditional algorithm and the new multi-agent algorithm were used to simulate a problem many times with a large number of randomly generated sets of initial conditions. Two parameters were varied: the total number of agents and the integration error tolerance. It was shown that the predicted relationship between the number of agents and the computational cost savings was valid and that requiring finer simulation accuracy serves to amplify the potential savings associated with the new algorithm.

REFERENCES

ALUR, R., BELTA, C., IVANCIC, F., KUMAR, V., MINTZ, M., RUBIN, H., AND SCHUG, J., PAPPAS, G. 2001. Hybrid modeling of biomolecular networks. *Hyb. Syst. Computation and Control*.

ALUR, R., GROSSE, R., HUR, Y., KUMAR, V., AND LEE, I. 2000. Modular specification of hybrid systems in charon. *Hybrid Systems Computation and Control: Third International Workshop 3*, 6–19.

ASCHER, U. AND PETZOLD, L. 1998. *Computer methods for ordinary differential equations and differential-algebraic equations*. Society for Industrial and Applied Mathematics, Philadelphia, Pa.

BAHL, V. AND LINNINGER, A. 2001. Modeling of continuous–discrete processes. In *Hybrid Systems: Computation and Control*, M. D. Benedetto and A. Sangiovanni-Vincentelli, Eds. Lecture Notes in Computer Science, vol. 2034. Springer-Verlag, New York, 387–402.

BRANICKY, M. 1995. Studies in hybrid systems: Modeling, analysis and control. Ph.D. dissertation. MIT, Cambridge, Mass.

CAROTHERS, C., PERUMALLA, K., AND FUJIMOTO, R. 1999. Efficient optimistic parallel simulation using reverse computation. *ACM Transactions on Modeling and Computer Simulation 9*, 3, 224–253.

CELLIER, F. 1979. Combined discrete/ continuous system simulation by use of digital computers: techniques and tools. Ph.D. dissertation. ETH Zurich, Zurich, Switzerland.

DESHPANDE, A. G. AND SEMENZATO, L. 1995. Shift programming language and runtime system for dynamic networks of hybrid automata. *California PATH project technical report*.

ENGSTLER, C. AND LUBICH, C. 1996. Multirate extrapolation methods for differential equations with different time scales. *Computing 58*, 173–185.

ESPOSITO, J. AND KUMAR, V. 2001. Efficient dynamic simulation of robotic systems with hierarchy. In *IEEE International Conference on Robotics and Automation* (Seoul, Korea). IEEE Computer Society Press, Los Alamitos, Calif., 2818–2823.

ESPOSITO, J., PAPPAS, G., AND KUMAR, V. 2001a. Multi-agent hybrid system simulation. In *IEEE Conference on Decision and Control* (Orlando, Fla), IEEE Computer Society Press, Los Alamitos, Calif.

ESPOSITO, J. M. 2002. Simulation and control of hybrid systems with applications to mobile robotics. Ph.D. dissertation. University of Pennsylvania, Philadelphia, Pa.

ESPOSITO, J. M., PAPPAS, G. J., AND KUMAR, V. 2001b. Accurate event detection for simulating hybrid systems. In *Hybrid Systems : Computation and Control*, M. D. Benedetto and A. Sangiovanni-Vincentelli, Eds. Lecture Notes in Computer Science, vol. 2034. Springer-Verlag, New York, 204–217.

FIERRO, R., DAS, A., SPLETZER, J., ESPOSITO, J., KUMAR, V., OSTROWSKI, J. P., TAYLOR, C. J., HUR, Y., ALUR, R., LEE, I., GRUDIC, G., AND SOUTHOLD, B. 2002. A framework and architecture for multi-robot coordination. *Inte. J. Robot. Automat. 21*, 10-11 (Oct.–Nov.), 2407–2411.

FUJIMOTO, R. 2000. *Parallel and distributed simulation systems*. Wiley Interscience, Cambridge, Mass.

GEAR, C. AND OSTERBY, O. 1984. Solving ordinary differential equations with discontinuities. *ACM Trans. Math. Softw. 10*, 1, 23–44.

GEAR, C. AND WELLS, D. 1984. Multirate linear multistep methods. *BIT 24*, 484–502.

HUR, Y. AND LEE, I. 2002. Distributed simulation of multi-agent hybrid systems. In *Proceedings of the 5th IEEE International Symposium on Object-Oriented and Real-Time Distributed Computing*, (Arlington, Va.). IEEE Computer Society Press, Los Alamitos, Calif., 356–364.

JEFFERSON, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst. 7*, 3 (July), 404–425.

LUBACHEVSKY, B. 1989. Efficient distributed event-driven simulations of multiple-loop networks. *Commun. ACM 32*, 1, 99–113.

MALER, O. AND PNUELI, A., Eds. 2003. *6th International Workshop, Hybrid Systems: Computation and Control* (Prague, Czech Republic). Lecture Notes in Computer Society, vol. 2623. Springer-Verlag, New York.

MIRTICH, B. 2000. Timewarp rigid body simulation. In *SIGGRAPH*. ACM, New York.

MOSTERMAN, P. 1999. An overview of hybrid simulation phenomena and their support by simulation packages. In *Hybrid Systems : Computation and Control*, F. Vaandrager and J. H. van Schuppen, Eds., Lecture Notes in Computer Science, vol. 1569. Springer-Verlag, New York, 163–177.

NICOL, D. M. AND PERRONE, L. F. 2000. Cost/benefit analysis of interval jumping in wireless power control simulation. In *Winter Simulation Conference*.

PARK, T. AND BARTON, P. 1996. State event location in differential-algebraic models. *ACM Trans. Model. Comput. Simulat. 6*, 2, 137–165.

SHAMPINE, L., GLADWELL, I., AND BRANKIN, R. 1991. Reliable solution of special event location problems for ODEs. *ACM Trans. Math. Softw. 17*, 1 (Mar.), 11–25.

STEINMAN, J. 1992. SPEEDED: A unified framework to parallel simulation. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*. 75–83.

TOMLIN, C., PAPPAS, G. J., AND SASTRY, S. 1998. Conflict resolution for air traffic management: A study in multi-agent hybrid systems. *IEEE Trans. Automat. Cont. 43*, 4.

Received July 2003; accepted March 2004