On-line Symbolic Constraint Embedding for Simulation of Hybrid Dynamical Systems

R. Brent Gillespie, Volkan Patoglu and Islam I. Hussein
Department of Mechanical Engineering, University of Michigan, 2350 Hayward Ave, Ann Arbor, MI 48109, USA

E. R. Westervelt
Department of Mechanical Engineering, The Ohio State University, 650 Ackerman Rd, Suite 255, Columbus, OH 43202, USA

Abstract. In this paper we present a simulator designed to handle multibody systems with changing constraints, wherein the equations of motion for each of its constraint configurations are formulated in minimal ODE form with constraints embedded before they are passed to an ODE solver. The constraint-embedded equations are formulated symbolically according to a re-combination of terms of the unconstrained equations, and this symbolic process is undertaken on-line by the simulator. Constraint-embedding undertaken on-the-fly enables the simulation of systems with an ODE solver for which constraints are not known prior to simulation start or for which the enumeration of all constraint conditions would be unwieldy because of their complexity or number. Issues of drift associated with DAE solvers that usually require stabilization are sidestepped with the constraint-embedding approach. We apply nomenclature developed for hybrid dynamical systems to describe the system with changing constraints and to distinguish the roles of the forward dynamics solver, a collision detector, and an impact resolver. We have developed a MATLAB® toolbox for symbolically generating equations of motion using Kane's method and prototyped a simulator with on-line constraint embedding and demonstrated the design in three representative examples.

Keywords: Symbolic Manipulation, Changing Constraints, Constraint Embedding, Hybrid Dynamical Systems

This paper has not been submitted elsewhere in identical or similar form, nor will it be during the first three months after its submission to *Multibody System Dynamics.*

1. Introduction

Various methods are available for the production of the equations of motion for constrained multibody systems, and each method typically produces the equations in a particular form. While the equations are equivalent in the sense that they possess the same solution, the form of the equations usually dictates which of several available numerical methods may be applied to obtain that solution. While the following classification is not exhaustive, let us identify three major families for the production and solution of the equations of motion for constrained multibody systems (see [14] for a similar classification): 1) constraint appending using the Method of Lagrange Multipliers, which produces differential algebraic equations (DAEs) for solution by a DAE solver, 2) projecting the unconstrained equations through a matrix produced numerically, which produces DAEs or ODEs requiring stabilization, and 3) symbolic constraint embedding, which produces ODEs in reduced form (containing only those variables associated with the degrees of freedom of the constrained system) that do not require stabilization.

Basically a whole spectrum of techniques is available to account for the effects of constraints: from the numerical Lagrange multiplier and projection methods to the symbolic constraint embedding methods. The constraint embedding approach is appealing because it permits one to apply an ODE solver, which generally requires less expertise than a DAE solver. The constraint-embedded equations are fewer in number and have fewer unknowns than their constraint-appended counterparts, which can lead to compact and numerically efficient equations, though such an outcome also requires the judicious use of generalized speeds (quasi-velocities) [25] [17].[1] On the other hand, DAE solvers offer a solution to the constrained system dynamics without requiring the

[1] The reduced equations are often dismissed as less numerically efficient [14], [7], especially given the availability of sparse matrix techniques for the constraint-

symbolic operations involved in eliminating dependent variables. One simply needs to invoke a high index DAE solver or negotiate the options available for index reduction and stabilization to ensure that the DAE solver provides a solution that satisfies the requirements of a given application [7].

In addition to relationships between the formulation method and the resulting form of the equations, one can consider whether the equations are particularized to a given multibody system description during compile-time or during run-time (simulation). Generally speaking, each formulation method has been developed with an eye toward application either during compile-time or during run-time but not both. For example, Kane's method is generally applied to build customized equations during compile-time, using computerized symbolic algebra. The Newton-Euler approach, on the other hand, is often used to particularize general-purpose equations to the given system description at run-time, setting numerical values to appropriately couple various force and moment balances. Aside from choosing a set of generalized coordinates, Lagrange's equations are also often particularized to a given problem at run-time. While the various needs of computational efficiency and flexible system description have driven the development of each method toward application either during compile-time or run-time in the past, new applications are appearing that motivate a re-investigation into the advantages available from each method and their suitability for application symbolically or numerically.

Now, in certain multibody systems, changes in constraint condition are likely to occur during the time period in which the solution of the equations of motion is desired. Such systems are said to have changing topology [11], [37] or intermittent motion [12]. Choosing a solution method and concomitantly, choosing a formulation method requires

appended equations. However, the judicious use of generalized speeds is not usually considered in such estimations.

special consideration for such systems. Also, whether the constraint descriptions are available prior to run-time may be critical to the choice of equation production and solution method.

Within the constraint-appending and numerical projection methods, approaches to handling changing constraints are readily available. The core unconstrained equations remain untouched through the change in constraint, so changes may be reflected simply by swapping the constraint equations that are appended or imposed numerically during each epoch of the simulation [12]. Even constraints that first appear during run-time can be handled, since the appropriate algorithms are essentially components of the numerical simulator. Within the constraint-embedding approach, on the other hand, handling changing constraints is a little more delicate. Nominally, the entire set of equations requires re-formulation upon each change in constraint condition. If all constraint conditions can be enumerated prior to simulation, then quite plausibly the various constrained system formulations could be linked together at run-time to form a hybrid dynamical system [10]. Then the constraint conditions active at any given time in simulation may be driven by run-time variables. However, constraint conditions that appear for the first time during run-time cannot be accommodated. If changes in constraint condition that arise during run-time are to be handled using the constraint embedding approach and the concomitant advantages are to be enjoyed, including numerical efficiency and use of an ODE solver without stabilization, then a means to symbolically embed constraints during run-time must be developed.

Examples of systems in which constraints cannot be fully described before run-time include shared gaming environments, where objects passed between users or between a user and a software agent are subject to on-line modifications of their composition or surface shape. Another example is a design environment in which multiple users collaborate on the specification of a virtual artifact. Also, retaining the description of a constraint until run-time allows data-hiding in a distributed simulation environment. For example, the parameterization of the shape of an object may be hidden from the simulation routine until the constraint is actually imposed. Data hiding allows libraries of constrained multibody systems to be developed and maintained independent of their function within simulations. Another class of applications that motivates on-line constraint handling lies in systems for which the enumeration of all constraint conditions prior to simulation would be unwieldy because of their complexity or number. On-line constraint handling would allow only those constraints to be considered that are actually encountered during simulation.

A specific example in which constraints arise during run-time is a multi-player video game, played on two or more networked computers. Two or more users who interact with the same dynamical system may form a kinematic chain whose equations are stiff relative to a simulation rate constrained by the network. As an alternative to a shared solution constrained by a slow network link, each computer may simulate the dynamics of its nearest neighbors, perhaps including models of the linked users. The dynamical equations or constraint equations would be transmitted across the network in symbolic form, and possibly with on-the-fly updates. The local dynamical equations could be combined with the constraint equations to form a local constrained dynamical description. State or parameter errors could then be communicated to ensure appropriate interaction.

In addition to changing contact conditions, the traversal of the solution through or near to a representation singularity motivates the consideration of modifying system equations on-the-fly. In this case, the modifications needed pertain to the choice of independent configuration and/or motion variables. For certain mechanisms (*i.e.* robots,) it may prove difficult to define a single set of independent motion variables suitable for all regions of the coordinate space. In such cases, a change in the choice of independent motion variables may be undertaken during simulation to avoid break-down of the solution method. Like constraint handling, this process may be undertaken either numerically or symbolically, but invariably a run-time process is needed. Again, the DAE approach essentially avoids this problem since it does not rely on a model in independent coordinates and is essentially run-time implemented. Use of the constraint embedding approach to avoid representation singularities by changing coordinates would require the development of constraint embedding routines that work on-line. That is, they would require the incorporation of symbolic routines into the run-time simulator code.

In fact, several recent innovations have enabled the development of new approaches to multibody code formulation and solution. Namely, fast computerized symbolic algebra is available and interpreted languages such as MATLAB allow combinations of symbolic and numerical operations to be undertaken using ad hoc and iterative code prototyping approaches. The opportunity has arisen to blur the distinction between symbolic compile-time and numerical run-time and breakdown traditional tradeoffs between formulation approach and solution approach. In this paper, we propose to incorporate symbolic manipulation routines into the run-time solution code. This might seem like an unlikely approach, given the high computational demands of symbolic routines and the very distinct heritage of symbolic and numerical methods. However, that is precisely what we have undertaken in our work and demonstrate in this paper.

We propose to apply computerized symbolic algebra not only to the process of formulating equations of motion, but also to the process of simulation. Our aim is to expand the tools available for simulation of systems with changing topology to the point where they will reflect the variety of options available for constant topology systems. We anticipate applications in which the relative simplicity and stability of ODE solvers are desired and the ability is required to impose the effects of constraints whose specifications are not available prior to run-time. We also anticipate the availability of sufficient computational power to invoke the symbolic operations necessary for eliminating dependent coordinates within a single time-step in a constant step-size solver. This would enable hardware- or human-in-the-loop simulation for systems with changing constraints, where the particular constraint conditions in effect are driven by a human or piece of hardware linked to the simulated dynamics through a computer interface.

Likewise, by incorporating symbolic routines into the run-time simulator code, representation singularities can be avoided using a change in coordinates, yet simple ODE solvers applied to equations in reduced (constraint embedded) form. Detection of the suitability of the active set of independent coordinates can be accomplished using a check of the condition number of the constraint Jacobian, and a new set can be chosen and re-formulation undertaken to avoid singularities, for example as described in [29].

We are particularly interested in the use of on-line constraint embedding to create virtual environments for exploration by a human user through a haptic interface. Our simulator is intended to facilitate interaction with mechanisms that include escapements, stops, and latches specified by a geometric model that retains significant independence from the dynamic solution engine. Our simulator treats collisions and transitions between free, rolling, and sliding motion, all with an ODE solver. We aim to reproduce the effects of making, possibly maintaining, and breaking contact between bodies, knowing that these effects are often incited by a user exploring a virtual environment. Naturally, interaction with a user in a virtual environments requires real-time simulation.

We have prototyped a simulator with on-line constraint embedding in MATLAB®. Development of the equations of motion is based on

Kane's method and the on-line re-formulation in independent coordinates is based on a re-combination of terms in the unconstrained equations according to the algorithm outlined in [36] and further extended in [25]. Although our simulator is currently not capable of real-time constraint embedding within a single time-step in a fixed-step solver, we do demonstrate embedding on-the-fly that runs in parallel with simulation and that is initiated a short period before the new constrained equations are needed.

In the following sections, we further review the field of constrained system formulation and simulation, contrasting on-line constraint embedding to existing methods. Thereafter, we develop nomenclature from the field of hybrid dynamical systems that we use to lay out our simulator design. Finally, we demonstrate the simulator in three examples and conclude.

2. Dynamical System Modelling

2.1. Kinematical and Dynamical Differential Equations

Consider a multibody system S whose configuration is described by n generalized coordinates q_r $(r = 1, \dots, n)$. To enable the formulation of compact and efficient equations of motion [31], define n generalized speeds u_r $(r = 1, \dots, n)$ as linear combinations of the generalized coordinate derivatives $\dot{q}_r$ $(r = 1, \dots, n)$ [17]. Express these definitions using

$$\mathbf{u} \triangleq Y\dot{\mathbf{q}} + Z, \tag{1}$$

where $\mathbf{u}$ and $\dot{\mathbf{q}}$ are $n \times 1$ matrices of u_r and $\dot{q}_r$, and where the elements of the $n \times n$ matrix Y and $n \times 1$ matrix Z are functions of q_r $(r = 1, \dots, n)$ and possibly time t. The matrices Y and Z in Eq. (1) must be chosen by the analyst in such a way that the reciprocal relations exist in which the generalized coordinate derivatives are expressed in terms of the generalized speeds:

$$\dot{\mathbf{q}} = W\mathbf{u} + X, \tag{2}$$

where the elements of the $n \times n$ matrix W and $n \times 1$ matrix X are functions of q_r $(r = 1, \ldots, n)$ and possibly time t. Equation (2) is a matrix arrangement of the kinematical differential equations, and forms the first of two portions of the state equations or equations of motion that govern the behavior of system S.

The second portion, the dynamical differential equations, may be expressed in matrix form as

$$M(\mathbf{q})\dot{\mathbf{u}} = \mathbf{f}(\mathbf{q}, \mathbf{u}, t) \tag{3}$$

which may be derived, for example, using Kane's method. In Kane's method, one carries out dot products between partial velocity vectors and applied and inertia forces, and between partial angular velocity vectors and applied and inertia torques. The partial velocity and partial angular velocity vectors are obtained by inspection of the pertinent velocity and angular velocity expressions, identifying coefficients of the corresponding generalized speeds. Then, a summation of terms over all particles and bodies in S produces expressions for the generalized active force F_r $(r = 1, \ldots, n)$ and the generalized inertia force F_r^* $(r = 1, \ldots, n)$. The dynamical differential equations are then contained in $F_r + F_r^* = 0$ $(r = 1, \ldots, n)$, which may be arranged as in Eq. (3).

If there are no configurations of the system for which the motion of one or more bodies is not resisted by inertia, then the mass matrix M is nonsingular and the equations of motion can also be expressed in explicit form as

$$\dot{\mathbf{u}} = \mathbf{F}(\mathbf{q}, \mathbf{u}, t) \tag{4}$$

where $\mathbf{F}$ is a $n \times 1$ matrix of generalized applied and inertia force terms. In this paper, we will focus on systems for which the mass matrix is nonsingular, since this is the case for the majority of systems of engineering interest. We refer readers to Chapter 6 in [30] for a detailed treatment of the singular case.

2.2. Constraint Equations

System S may be subject to constraints, including constraints that act only during a time segment within the time interval of interest. Suppose there are l configuration constraints given as

$$\Phi(\mathbf{q}) = 0. \tag{5}$$

In certain formulations, to be reviewed in the next section, configuration constraints are treated as motion constraints by differentiating them with respect to time:

$$\Phi_q \dot{\mathbf{q}} = 0. \tag{6}$$

In place of imposing the configuration constraints within the solver (say, by performing Newton iterations on Eq. (5) at each time step), the configuration can be found by integrating the motion constraints. However, numerical or initialization errors in the motion constraints will lead to violations of the configuration constraints –violations that can accumulate and slow down the integration over long simulations. Since the initialization and round-off errors at the motion (velocity) level will remain relatively constant during integration [14], the violation to the configuration constraints will only grow linearly with time. There exist simple stabilization techniques to overcome this problem [31].

Let there be an additional $m - l$ motion constraints, where $l \leq m \leq n$. Suppose further that the motion constraints are linear in the u_r, so that after the application of Eq. (2), the differentiated configuration constraints can be combined with the motion constraints, yielding

$$B\mathbf{u} + C = 0, \tag{7}$$

where the elements of the $m \times n$ matrix B and the $m \times 1$ matrix C are functions of q_r $(r = 1, \ldots, n)$ and possibly time t.

Some of the index reduction formulations require that the constraints (7) be differentiated to the acceleration level

$$B\dot{\mathbf{u}} + \dot{B}\mathbf{u} + \dot{C} = 0, \tag{8}$$

so that they can be solved together with the dynamical differential equations.

Care should be taken when imposing acceleration level constraints instead of their motion-level or configuration level counterparts since in this case the drift phenomenon is critical. The initialization and numerical errors that occur at the acceleration level will lead to motion-level violations that grow linearly and configuration-level violations that grow quadratically in time. Therefore, it is always good practice to use stabilization with these formulations.

Equations (4) together with configuration and motion constraints form a system of n first-order differential and m algebraic equations to be solved for the n generalized speeds u_r. Note that the n first-order kinematical differential equations (2) are typically solved alongside Eqs. (4) and (7) to determine the generalized coordinates q_r.

Although we have identified more equations than unknowns, the system is in fact not overdetermined, since associated with the constraint equations are constraint forces that restrict the motion of S. There will be m constraint force components, one for each degree of freedom that is restricted. Let us call the constraint force components ξ_s $(s = 1, \ldots, m)$.

There are three general approaches to the analysis of constrained dynamical systems as discussed in Section 1. The first employs the Method of Lagrange Multipliers, the second a numerical projection matrix R, and the third is called embedding the constraints. The next section discusses these formulations along with their implementation in simulators to handle systems with changing constraints.

3. Approaches to the Formulation and Simulation of Systems Subject to Changing Constraints

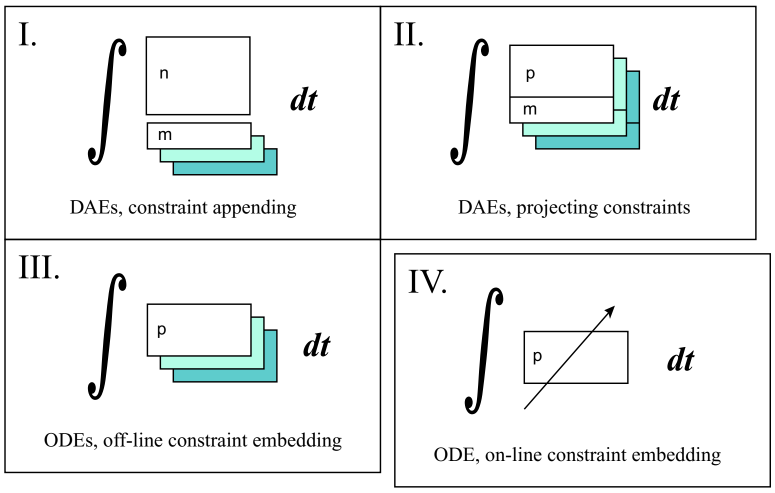
I.
n
m
dt
DAEs, constraint appending
II.
p
m
dt
DAEs, projecting constraints
III.
p
dt
ODEs, off-line constraint embedding
IV.
p
dt
ODE, on-line constraint embedding

Figure 1. Four designs for a simulator based on I.) The Method of Lagrange Multipliers, II.) Projection Matrices, III.) Off-line Constraint Embedding, and IV.) On-line Constraint Embedding (introduced in this paper). Parameter n stands for the number of generalized speeds, m stands for the number of algebraic constraints and $p \triangleq n - m$. Stacked and shaded boxes indicate the manner in which each approach may be adapted to treat systems with changing constraints.

Figure 1 shows schematically four possible designs for an interactive simulator capable of handling systems subject to changing constraints.[2]

[2] There exist other alternatives that will not be considered in this paper. One of them is based on computing the constraint forces explicitly as the solution of a linear complementarity problem and applying them to the unconstrained system of equations [2] [23]. Satisfaction of the constraints then follows by the action of the constraint forces, through the dynamical analysis.

This section will be organized around Figure 1. The information contained in each schematic representation is twofold. First, the labelled boxes indicate the number of dynamical differential equations and algebraic constraint equations and make the distinction between the use of the constraints to eliminate dependent coordinates and their use to augment the analysis. Secondly, the arrangement of shaded boxes depicts how each simulator is cast to handle changing constraints.

Design I is based on the Method of Lagrange Multipliers. After the unconstrained system of equations is developed, the Method of Lagrange Multipliers is used to append the constraint equations, by introducing m multipliers λ_s $(s = 1, \ldots, m)$ and adding the term $B^T\lambda$ to Eq. (3) leading to

$$M(q)\dot{\mathbf{u}} = \mathbf{f}(\mathbf{q}, \mathbf{u}, t) + B(\mathbf{q})^T\lambda \tag{9}$$

where λ is a $m \times 1$ column matrix of undertermined multipliers. The multipliers λ may be identified with the constraint force components [13]. If no motion constraints are imposed, Eqs. (9) may be solved together with (5) as a DAE of index 3. Although index-3 solvers are available [26], the algebraic constraint equations are usually differentiated once or twice to reduce their index and thus prepare the equations for the more readily available index-2 or index-1 DAE solvers. Solving Eq. (9) together with (7) produces a DAE of index 2 or solving with equation (8) produces an index-1 DAE. Both index-1 and index-2 formulations require stabilization to the constraint manifold since information is lost when differentiating the constraint equations. Baumgarte stabilization [4] and the Augmented Lagrangian formulation [5] are among the numerous stabilization methods available for index-1 formulations. A stabilized formulation also exists for index-2 DAEs [8].

For the treatment of systems with changing constraints using the simulator architecture of Design I, the core n equation Eqs. (9) remain the same throughout the simulation (except for the constraint Jacobian

$B(q)$, which reflects only the active constraints). The m active algebraic constraint equations are applied to the integration and may change during simulation. The simulator designs described in [12] and [20] fit Design I. In [12, 38], Haug et.al. demonstrated the management of constraints within the Method of Lagrange Multipliers and the use of DAE solvers.

Design II makes use of numerical projection methods. The n unconstrained equations (9) are projected onto the $p \stackrel{\Delta}{=} n-m$ dimensional constraint manifold by pre-multiplying with a $p \times n$ projection matrix R, which is produced numerically during solution from the constraint Jacobian B via an SVD [34], QR decomposition [19], or Gauss triangularization [33]. The projection matrix R satisfies the equation $R^T B^T = 0$, and the projected dynamical differential equations are given by

$$R^T M \dot{\mathbf{u}} = R^T f. \tag{10}$$

These numerically projected equations require simultaneous solution of the constraint equations, so that together with the constraint equations, the problem may be formulated as an index-2 DAE of $p + m = n$ equations. Twice differentiating the configuration constraints produces an ODE, although its solution again requires stabilization.

For the treatment of systems with changing constraints, the simulator architecture of Design II features p dynamical equations produced by numerical projection. The m algebraic constraint equations are solved simultaneously, and the dynamical equations must be projected using a new projection matrix each time the constraints change. Adaptation of the projection methods to handle changing constraints is relatively easy since whenever a change in constraints occurs: only the projection matrix R is affected due to the change in constraint Jacobian B; however, R is already generated on the fly numerically.

In **Design III** the configuration constraints are differentiated and grouped with the motion (nonholonomic) constraints to produce constraint equations that are linear in the motion coordinates, as in Eq. (7). These constraints are then used to undertake a local coordinate transformation to eliminate dependent motion variables from the equations [17]. This process projects the dynamical differential equations onto the p-dimensional constraint manifold, in which the solution of (4) is constrained to lie. But here the projection is carried out using symbolic operations (realizing a dot product) in contrast to numerical operations and is called *embedding* the constraints. For a geometrical interpretation of the projection operation carried out in Kane's method, see [21] or [9].

Thus for treating systems subject to changing constraints, simulator Design III features switching among a set of ODEs, each produced by embedding the pertinent constraints. Since the constraints are embedded, there are only p dynamical equations to solve for the system in each of its constraint conditions where p may vary by condition. However, in Design III, all of the constraints must be known prior to the time of simulation. Such an approach is often suitable for the simulation of mechanisms, as described in [10].

In **Design IV**, the configuration constraints are differentiated and again grouped with the motion constraints and used to eliminate dependent motion variables as in Design III. Again, the embedding process yields only p dynamical differential equations for the system in each of its constraint conditions. However, unlike Design III, the constraints are embedded on-line (during simulation) through symbolic manipulation of the dynamical equations. Thus, the equations resident in the integrator are not simply swapped in and out; they are subject to reformulation on the fly. The incorporation of on-line formulation into the simulator design allows updates to the geometric model that occur during simulation to be reflected in the behavior of the dynamic model. The simulator architecture depicted in Design IV is the new scheme proposed in this paper.

Whether embedding the constraints is undertaken at compile-time or at run-time, to embed in Kane's method, one expresses the m dependent generalized speeds in terms of the remaining p independent generalized speeds by carrying out linear operations on the constraint equations (7). One may begin by re-ordering and partitioning the generalized speeds in (7) to produce:

$$B_1 \begin{bmatrix} u_{p+1} \\ \vdots \\ u_n \end{bmatrix} = B_2 \begin{bmatrix} u_1 \\ \vdots \\ u_p \end{bmatrix} + C. \tag{11}$$

where u_r $(r = 1, \ldots, p)$ are the independent generalized speeds and u_r $(r = p+1, \ldots, n)$ are the dependent generalized speeds. Further, one must choose the B_1 and B_2 matrices so that Eq. (11) may be solved uniquely for the dependent generalized speeds and the results written in terms of $D = B_1^{-1}B_2$ and $E = B_1^{-1}C$ as:

$$\begin{bmatrix} u_{p+1} \\ \vdots \\ u_n \end{bmatrix} = D \begin{bmatrix} u_1 \\ \vdots \\ u_p \end{bmatrix} + E, \tag{12}$$

where the elements of D and E are functions of q_i $(i = 1, \ldots, n)$ and possibly time t.

Now that the motion constraints have been expressed in an explicit linear form, the derivation of the constrained dynamical differential equations may proceed according to one of two methods. In the first, Eq. (12) may be used to eliminate the dependent generalized speeds from the analysis and Kane's method applied as usual. Alternatively, the constrained dynamical equations can be obtained by a re-combination of terms within the unconstrained dynamical equations. Specifically, the constrained generalized active force $\tilde{F}_r$ $(r = 1, \ldots, p)$ and the constrained generalized inertia force $\tilde{F}_r^*$ $(r = 1, \ldots, p)$ may be formed from the unconstrained generalized active force F_r $(r = 1, \ldots, n)$ and generalized inertia force F_r^* $(r = 1, \ldots, n)$ as follows:

$$\tilde{F}_r = F_r + \sum_{s=p+1}^{n} D_{sr} F_s \quad (r = 1, \ldots, p) \tag{13}$$

$$\tilde{F}_r^* = F_r^* + \sum_{s=p+1}^{n} D_{sr} F_s^* \quad (r = 1, \ldots, p) \tag{14}$$

where D was defined in Eq. (12). To eliminate the dependent generalized speeds and dependent generalized speed derivatives from the resulting expressions, one may substitute from the motion constraints (Eq. (12)) and differentiated motion constraints [25] [36]. (See also Eqs. (4.4.3) and (4.11.4) in [17]).

The equations of motion in the independent generalized speeds are then simply formed as:

$$\tilde{F}_r + \tilde{F}_r^* = 0 \quad (r = 1, \ldots, p) \tag{15}$$

which are only p ordinary differential equations in the p unknowns, u_r $(r = 1, \ldots, p)$. Finally, equations (15) may be used to produce explicit equations for $\dot{u}_r$ in the form

$$\dot{\tilde{\mathbf{u}}} = \tilde{\mathbf{F}}(\mathbf{q}, \tilde{\mathbf{u}}, t) \tag{16}$$

where $\tilde{\mathbf{u}}$ is a $p \times 1$ matrix of the independent generalized speeds.

The resulting dynamic differential equations are a set of ordinary differential equations, and yield to solution with a standard ODE solver. Advantages associated with embedding constraints include the reduction in the number of equations to be integrated and robustness due to the disappearance of the instability problem associated with the integration of differentiated constraints in DAE solvers.

The process outlined above for obtaining equations of motion is aided by the use of symbolic manipulation software. One such symbolic package is Autolev [32] [16] and another is SymBody, a new toolbox for MATLAB® created by the authors and described in Section 5.5 below.

The recombination of terms prescribed in Eqs. (13) and (14) are carried out during simulation using computerized symbolic algebra to produce the equations in minimal form for the system in each of its constraint conditions, just before they are needed by the numerical solver. Before demonstrating the process, we first introduce some notation for hybrid dynamical systems, and then describe its incorporation with two additional simulator components: a collision detector and impact resolution algorithm.

In the following, we review the language of hybrid dynamical systems and apply it to the formulation of a simulator to handle the interacting continuous-time and discrete-time dynamics. The intention is to capture both the “memory” in the continuous system dynamics and the “memory” in the discrete dynamics and their interaction to create a system whose behavior accurately reflects the behavior of its target system.

4. Hybrid System Modelling

To prepare for the construction of a simulator that can advance the solution of a constrained dynamical system through changes in the constraints, reflecting changes in contact condition, we borrow nomenclature and modeling tools from the field of Hybrid Dynamical Systems. Review papers in the field of hybrid dynamical systems include [1] and [6]. The purpose of this section is to present a model that can sequentially and interactively patch together various dynamical subsystems with appropriate initial and final states.

Hybrid dynamical systems are systems that exhibit interacting discrete state and continuous state dynamics. The term *interacting* indicates that changes in discrete states influence the evolution of the continuous dynamics and changes in continuous states influence evolution in the discrete dynamics. Such interaction precludes an analysis that treats the continuous and discrete models separately.

We shall concentrate here on hybrid dynamical systems in which the discrete and continuous dynamics interact only at discrete points in time known as *events*. This enables a description in terms of a single discrete state subsystem, a collection of continuous state subsystems, and a description of the possible interactions between the two subsystem classes. We will also restrict the discussion to systems in which the continuous state dynamics can be modelled in the form of ODEs or DAEs, defined on open subsets of the time interval of interest. That is, we employ a continuous time formulation, realizing that for numerical solution, these continuous dynamics will be approximated by a discrete time system. To describe the discrete subsystem, we employ a finite state machine. Together, the collection of continuous and discrete subsystems expressed in this form may be represented in a *hybrid automaton* [1] and [3]. A hybrid automaton, then, can be considered a finite state machine in which the discrete states have been replaced by co-called "modes", each of which indicate a particular continuous dynamics. Each mode is a description of a dynamical system that applied for a period of time that is governed by the discrete dynamics.

While the continuous state may change at any time, the state of the discrete subsystem changes only at the events. At these events, the discrete subsystem exerts its influence over the continuous dynamics in one or both of two ways: a) by causing a *switch* in the active differential equations describing the continuous state evolution, and/or b) by causing a *jump* or discontinuity in the continuous states.

The hybrid dynamics are abstracted into a collection $S = \bigcup_{k=1}^{n_m} S_k$, of *modes* S_k, where n_m is the number of modes and where the state $x^{(k)}$ is defined as the collection of kinematical variables and dynamical variables $[\mathbf{q} \quad \mathbf{u}]^T]$ defined for mode S_k. The state within mode S_k evolves according to the differential equation

$$\dot{\mathbf{x}}^{(k)} = \mathbf{F}^{(k)}(\mathbf{x}^{(k)}, \boldsymbol{\theta}^{(k)}, t), \tag{17}$$

where $\boldsymbol{\theta}^{(k)}(t)$ is an exogenous input for the k-th mode. The superscripts (k) index not only the functions $\mathbf{F}$ but also the state $\mathbf{x}$ and input $\boldsymbol{\theta}$ to allow different state and input variable definitions and possibly different state and input dimensions between modes. Note that for the simulation of mechanical systems, whose state equations are generally second order, the state $\mathbf{x}$ will comprise the generalized coordinates q_r $(r = 1, \ldots, n)$ and the generalized speeds u_r $(r = 1, \ldots, p)$.

Also associated with each mode S_k is a set of pending transitions J_k to other modes, where $j \in J^{(k)}$ is the index of the new mode following the event and $J^{(k)}$ is the set of modes reachable from the k-th mode. The timing of the events is determined by a *switching function*:

$$\mathbf{f}_{j,i}^{(k)}(\mathbf{x}^{(k)}, \boldsymbol{\theta}^{(k)}) = 0, \quad i = 1, \ldots, n_j^{(k)}, \quad j \in J^{(k)}, \tag{18}$$

where $n_j^{(k)}$ is the number of transitions from mode k to mode j.

The time t^* that, together with the state $\mathbf{x}^{(k)}(t^*)$ and specified motion $\boldsymbol{\theta}^{(k)}(t^*)$, produces a zero of switching function $\mathbf{f}_{j,i}^{(k)}$ is called a *switching instant*; it triggers the associated transition. Once a transition is triggered, an associated reset function $\boldsymbol{\phi}_{j,i}^{(k)}$ is executed to map the final state values $\mathbf{x}^{(k)}$ in the current mode to the initial state values $\mathbf{x}^{(j)}$ in the next mode.

$$\mathbf{x}^{(j)} = \boldsymbol{\phi}_{j,i}^{(k)}(\mathbf{x}^{(k)}, t^*), \quad i = 1, \ldots, n_j^{(k)}, \quad j \in J^{(k)} \tag{19}$$

A special reset function sets the initial conditions for the initial mode S_1 (The initial mode is arbitrarily labelled with $k = 1$).

$$\mathbf{x}^{(1)}(0) = \mathbf{x}_0^{(1)} \tag{20}$$

Evaluation of the hybrid system can be viewed as a sequence of subproblems, each characterized by a continuous evolution in a mode terminated by an event (*i.e.* zero crossing of a switching function), and then evaluation of the reset functions to initialize the new mode.

It is convenient to represent the interacting continuous and discrete dynamics of the hybrid system using an automaton as in Figure 2. This

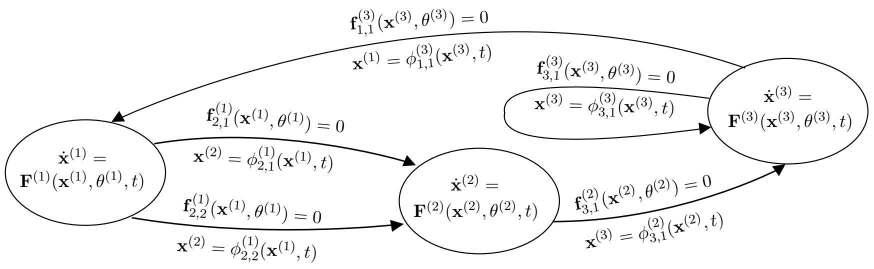
$\dot{\mathbf{x}}^{(1)} = \mathbf{F}^{(1)}(\mathbf{x}^{(1)}, \theta^{(1)}, t)$
$\dot{\mathbf{x}}^{(2)} = \mathbf{F}^{(2)}(\mathbf{x}^{(2)}, \theta^{(2)}, t)$
$\dot{\mathbf{x}}^{(3)} = \mathbf{F}^{(3)}(\mathbf{x}^{(3)}, \theta^{(3)}, t)$
$\mathbf{f}^{(3)}_{1,1}(\mathbf{x}^{(3)}, \theta^{(3)}) = 0$
$\mathbf{x}^{(1)} = \phi^{(3)}_{1,1}(\mathbf{x}^{(3)}, t)$
$\mathbf{f}^{(3)}_{3,1}(\mathbf{x}^{(3)}, \theta^{(3)}) = 0$
$\mathbf{x}^{(3)} = \phi^{(3)}_{3,1}(\mathbf{x}^{(3)}, t)$
$\mathbf{f}^{(1)}_{2,1}(\mathbf{x}^{(1)}, \theta^{(1)}) = 0$
$\mathbf{x}^{(2)} = \phi^{(1)}_{2,1}(\mathbf{x}^{(1)}, t)$
$\mathbf{f}^{(1)}_{2,2}(\mathbf{x}^{(1)}, \theta^{(1)}) = 0$
$\mathbf{x}^{(2)} = \phi^{(1)}_{2,2}(\mathbf{x}^{(1)}, t)$
$\mathbf{f}^{(2)}_{3,1}(\mathbf{x}^{(2)}, \theta^{(2)}) = 0$
$\mathbf{x}^{(3)} = \phi^{(2)}_{3,1}(\mathbf{x}^{(2)}, t)$

Figure 2. An example hybrid automaton with three modes and five transitions.

simple hybrid automaton has 3 modes with two transitions from mode 1 to 2, one transition from mode 2 to 3, one transition from mode 3 to 1 and a self transition in mode 3.

5. Simulator Architecture

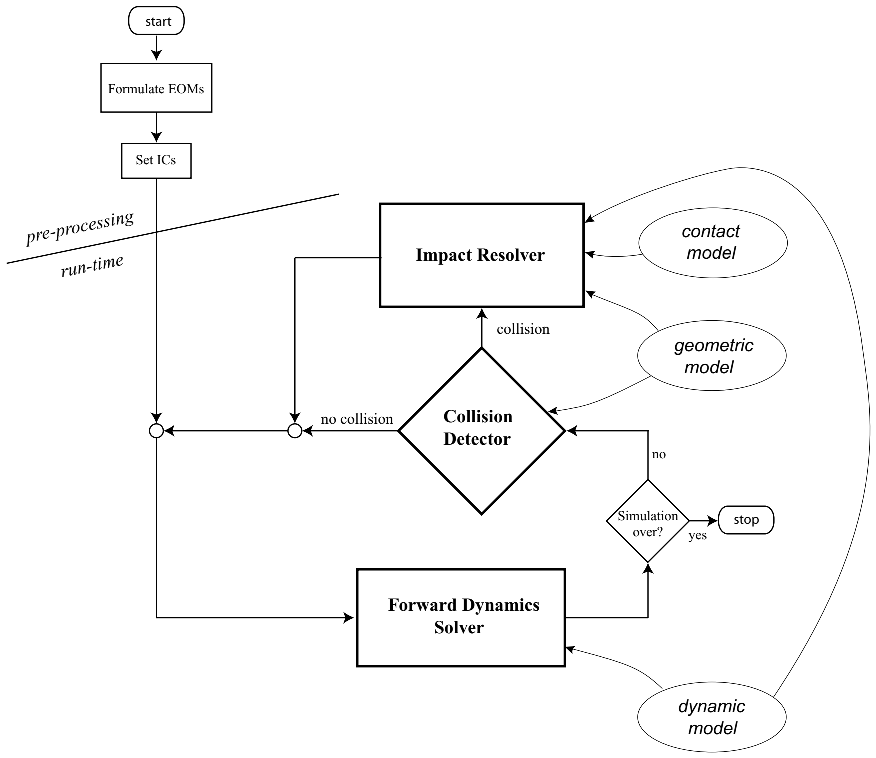

Figure 3. Standard Simulation Flow Chart.

A simulator equipped to handle multibody systems with changing constraint conditions requires three major components: a forward dynamics solver (an ODE or DAE solver), a collision detector, and a means for resolving impacts as indicated in the flow chart in Figure 3. The forward dynamics solver advances the solution of $\mathbf{F}^{(k)}$ in continuous time between events. The collision detector checks for contact between bodies by evaluating a collection of transition functions $\mathbf{f}_{j,i}^{(k)}$. (Additional transition functions are used to detect interaction forces that become tensile between unilaterally constrained bodies, to trigger loss of contact.) The impact resolver, triggered into action by the collision detector, computes reset functions $\phi_{j,i}^{(k)}$ to initialize the forward dynamics solver.

Alongside these three major simulator components, there exist three models for the system: a dynamic model, a geometric model, and a contact model. Roughly, the numerical integrator will maintain the dynamic model, the collision detector handles the geometric model, and the impact resolver calls upon the contact model. The impact resolver however, also consults the geometric model and possibly the dynamic model to compute the appropriate impulse response between bodies.

The use of a collision detector and interaction calculator, as in Figure 3, ensures that interacting bodies respond to each other's presence. Note that the impact resolver is called only intermittently whereas the collision detector and forward dynamics solver run continually, either alongside or subsequent to each other in computational time.

Let us now consider each of the elements of the flow chart in greater detail.

5.1. Forward Dynamics Solver

The forward dynamics solver operates on the equations of motion, a set of differential equations in the configuration and motion variables and inertia parameters. The multibody model can contain embedded or appended configuration or motion constraint equations written in terms of certain geometric parameters that are not necessarily part of the geometric model. In addition to new initial conditions from the impact resolver, the forward dynamics solver may respond to forces and moments applied during simulation, perhaps through the action of a human user interacting through a haptic interface.

5.2. Collision Detector

A collision detector is an algorithm that operates on a continually updated geometric model to determine points in time at which objects make contact with one another. At initial contact, the collision detector triggers the impact resolution algorithm that computes the interaction forces or impulses that act, in simulation, to prevent interpenetration of the two colliding objects.

A set of surface patches and their interconnection can be used to describe the geometry of each body in the simulation environment. The whole collection of surface patches along with its connected graph is called the geometric model.

There exists an extensive literature on the collision detection problem. For a detailed overview of existing methods for different geometric representations, we refer the reader to survey papers [15], [22]. In previous work [27], we have also contributed a collision detector that treats objects whose boundaries are represented using parametric surface patches. In [27] we presented a method for finding and tracking the closest points between two parametric surfaces based on a control problem formulation and the design of a stabilizing controller. The algorithm simultaneously accounts for the surface shape and motion while asymptotically achieving (and maintaining) the closest points.

Features of this approach include its guaranteed stability and seamless integration with the forward dynamics solver.

5.3. Impact Resolver

Impact resolution considers the problem of finding the separation velocities of two contacting bodies given the approach velocities and an appropriate contact model. Using Kane's method, the problem is formulated by expressing generalized impulse and momentum in terms of independent generalized speeds and then calculating the change in the generalized speeds under the assumptions dictated by a contact model. Independent of the contact model used, the impact resolution problem makes two assumptions: the configuration of the system does not change due to impact and the forces other than the action-reaction forces at the contact point can be ignored.

There exist several contact models in the literature [35], [18]. The contact model proposed by Smith [35] consists of assumptions that can be embodied in three equations about the impulse and relative momentum during impact. The first equation is provided by assuming that the components of the approach and separation velocities in the contact normal direction are related by a factor ϵ, called the coefficient of restitution. Two more equations can be formulated assuming that the tangential contact forces obey Coulomb's law of friction.

With these assumptions, the resolution model fully determines the response of the system to an impact. If the coefficient ϵ is non-zero, then the two bodies are guaranteed to rebound. Depending on whether the tangential forces are inside the friction cone or not, the two bodies will rebound in the direction of the contact normal at the contact point if inside the friction cone and in a direction determined by impulses that lie in the tangent plane if outside the friction cone. On the other hand, if $\epsilon = 0$, the two bodies will remain in contact. For a more detailed discussion of the contact and friction models, we refer the reader to [35] and the references therein.

5.4. Simulator Flow of Operation

We are interested in combining a forward dynamics solver, a collision detector, and an impact resolver to render the dynamics of a hybrid system using only an ODE solver, and in such a way that changes in the geometric model that occur on-line may be reflected in the simulated behavior. Thus our job is to perform on-line symbolic embedding of constraints. To lay out our architecture, we now elaborate on the flowchart in Figure 3, describing the incorporation of symbolic routines responsible for embedding constraints. Before presenting the elaborated flowchart, further perspective on our simulator can be developed by comparing Designs III and IV in Figure 1 in light of the example hybrid automaton shown in Figure 2. In fact, the hybrid automaton and the associated nomenclature in Section 4 is more suitably reflected in Design III. Each mode contains a particular pre-compiled (constraint embedded) dynamical model that will be passed to the integrator for simulation during the epoch in which it is active. In effect, the simulator switches among the pre-compiled models, as depicted in Figure 1, Design III. In this paper, however, we propose the use of on-line symbolic manipulation to produce the dynamical models in each of the automaton modes during run-time. One might think of each mode containing the same core equations $F_r + F_r^* = 0$ $(r = 1, \ldots, n)$, which, when the mode is actually visited, are then modified to produce the appropriate $\tilde{F}_r + \tilde{F}_r^* = 0$ $(r = 1, \ldots, p)$ that reflect the presence of the constraints identified with that mode. Using this on-line approach to formulating the constrained equations, it becomes possible to handle constraints whose full expression are not known at the simulation start.

Using on-line constraint embedding, the following steps must occur with each switching function zero-crossing: integration must be stopped; the reset functions invoked and, if a new mode is triggered reflecting the imposition of new constraints, then independent generalized speeds must be chosen and the dependent generalized speeds expressed in terms of the independent generalized speeds; the constrained dynamical equations formulated; and finally, integration re-started.

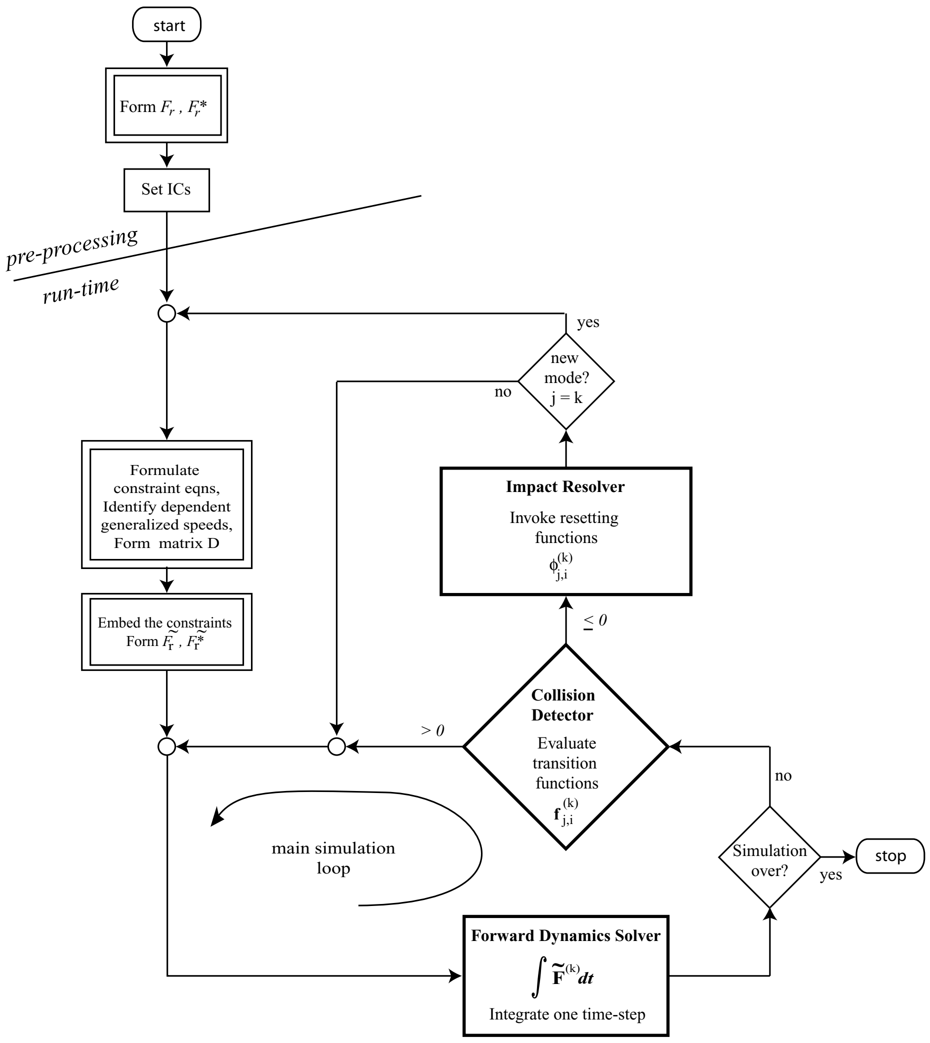
start
Form F_r , F_r^*
Set ICs
pre-processing
run-time
yes
new mode? j = k
no
Formulate constraint eqns, Identify dependent generalized speeds, Form matrix D
Impact Resolver
Invoke resetting functions $\phi_{j,i}^{(k)}$
Embed the constraints Form $\tilde{F}_r$, $\tilde{F}_r^*$
≤ 0
Collision Detector
Evaluate transition functions $\mathbf{f}_{j,i}^{(k)}$
> 0
no
main simulation loop
Simulation over?
yes
stop
Forward Dynamics Solver
$\int \tilde{\mathbf{F}}^{(k)} dt$
Integrate one time-step

Figure 4. Simulation Flow Chart for Design IV. Double frames indicate symbolic operations undertaken by SymBody.

To reflect all the operations involved in real-time constraint embedding, we present in Figure 4 an elaborated version of the flowchart of Figure 3. Consider traversing a path through the flowchart starting at the top. First, the unconstrained dynamical differential equations $F_r + F_r^* = 0$ $(r = 1, \ldots, n)$ are formulated and the initial conditions are set (Eq. (20)). Then, interactive simulation is ready to begin. A real-time clock is used to trigger the sampling of data from sensors and the issuing of commands through a digital-to-analog converter to actuators. The integration step must be traversed once each sample-time to update the behavior of virtual dynamical objects.

Before the equations are ready for integration, any active constraints must be embedded. The active constraint equations, expressed as dependencies among the generalized speeds in the form of Eq. (7) are collected. The generalized speeds are partitioned into sets of independent and dependent variables and the matrix D whose elements are D_{sr} is produced as in Eq. (12). Finally the constrained generalized active force $\tilde{F}_r$ and constrained generalized inertia force $\tilde{F}_r^*$ are formed using D_{sr} in Eqs. (13) and (14) and the constrained equations of motion are formed, $\tilde{F}_r + \tilde{F}_r^* = 0$ $(r = 1, \ldots, p)$. Formation of the constrained dynamical equations are all accomplished using symbolic algebra routines executed during run-time.

Simulation then proceeds within a particular mode by executing the main integration loop using $\tilde{\mathbf{F}}^{(k)}$ until any one of the transition functions $\mathbf{f}_{j,i}^{(k)}$ associated with mode k crosses zero. A switching function zero-crossing initiates an excursion from the main simulation loop: First, the associated reset function $\phi_{j,i}^{k}$ (if any) is invoked. If $j = k$ or there is no change in mode, then the process returns to the main simulation loop. If, however, a new mode is reached, then new constrained equations of motion are formulated symbolically. Figure 4, then, presents a simulation paradigm in which the equations to be integrated are ODEs in a set of independent variables, no matter what constraints might hold at a particular time. This enables the use of a standard ODE solver operating on dynamical equations in minimal form. As a result, the main simulation loop is as fast as possible and enjoys inherent stability. Of course if simulation is to take place in real-time, all loops must be traversable in a single time-step.

Using on-line symbolic manipulation and constraint embedding, an additional feature can be added to the simulator architecture depicted in Figure 4. Representation singularities, which exist in the configuration space of certain mechanisms, can be avoided with the incorporation of a check on the condition number of the constraint Jacobian within the main simulation loop. Such singularities can be side-stepped by triggering a re-selection of the independent generalized speeds using symbolic routines. For additional details, see [29] or [9].

To evaluate transition functions that trigger the deletion of constraints, expressions for interaction forces between bodies are needed. That is, expressions for the constraint forces are needed, which were eliminated from the analysis by constraint embedding. Expressions for the constraint forces can easily be produced, however, using the method of auxiliary generalized speeds [17] and such expressions will be uncoupled from the dynamical analysis. They are algebraic functions of the generalized coordinates, generalized speeds, and system parameters. The constraint forces can then be evaluated in the transition functions for the the purpose of, say, deleting a constraint when the interaction force becomes tensile and thereby realizing a unilateral constraint.

5.5. Symbolic Manipulation in MATLAB®

We have written a new MATLAB® toolbox for carrying out Kane's method, which we call SymBody. Like Autolev, a symbolic manipulator written in C for dynamic system analysis [32], SymBody provides tailored data objects to describe components and geometric entities such as bodies, points, forces and moments and their relationships with each other. For the implementation of the code, a new variable type called `timevars` and a symbolic time differentiation routine `dt` are defined in MATLAB®. Other core elements of the code are the definitions of vector variables and vector operations, most importantly the dot and cross products on vectors. For simplification of symbolic expressions and symbolic operations on scalars, matrices, and vectors, SymBody makes use of well developed MATLAB® Symbolic Toolbox and the Maple Kernel. The main motivation for re-creating Autolev inside MATLAB® was to leverage the high-level language and the powerful symbolic/numerical routines already available in MATLAB®.

Equations of motion for a dynamic system are automatically generated using Kane's method once necessary kinematic and dynamic variables are defined by the user using an input script. Constraint equations can be embedded into equations of motion to obtain the minimal representation using the `constrain` command. Since commands defined within a MATLAB® toolbox are available even to subroutines, newly defined symbolic operations can be easily incorporated into the routines responsible for numerical integration. We incorporate certain symbolic operations into the use of MATLAB® ODE solvers with event handling, as further described below.

6. Example 1

Figure 5 shows a planar system comprising a uniform disk B of radius r, mass m, and central moment of inertia J in contact with a ramp A inclined at an angle ϕ with the horizontal. Let N designate a Newtonian reference frame and let the unit vectors $\mathbf{a}_1, \mathbf{a}_2$, and $\mathbf{a}_3$ be directed as shown.

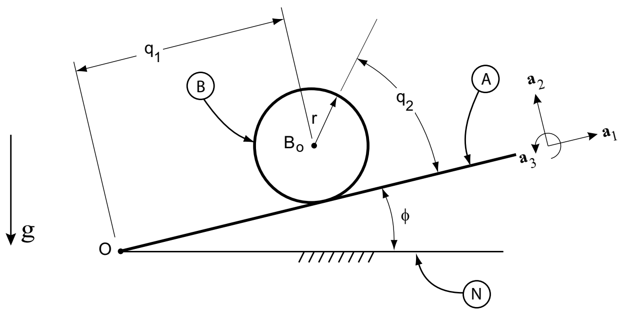

Figure 5. A rolling disk

To characterize the configurations in which B maintains contact with A, we may use the displacement q_1 between a fixed point O and the center B_o of B and the angle q_2 between the ramp edge and a line fixed in B that is initially parallel to the ramp edge. To characterize the motion of B in N, define generalized speeds as $u_1 \triangleq \dot{q}_1$ and $u_2 \triangleq r\dot{q}_2$.

If B slides on A, then both u_1 and u_2 are independent variables. However, if B rolls on A, a motion constraint may be written $u_1 + u_2 = 0$. When the rolling constraint is enforced, the equations may be written in terms of a single generalized speed (but still two generalized coordinates). The corresponding definition of the D matrix is: $D = [-1]$, which will be useful for on-line constraint embedding.

In preparation for the formulation of equations of motion for the disk in both the sliding and the rolling mode, we first formulate the unconstrained (sliding) equations. The forces acting on B, including the gravity force acting on the mass center B_o and the contact forces acting at the point of contact with A, may be resolved into a resultant $\mathbf{R}^B = -mg\mathbf{n}_2 - F_f\mathbf{a}_1 + \mathcal{N}\mathbf{a}_2$ applied at B_o and a torque $\mathbf{T}^B = -rF_f\mathbf{a}_3$, where F_f is the friction force, $\mathcal{N}$ is the normal force of contact, g is the local gravitational constant, and $\mathbf{n}_2$ is a unit vector directed vertically upward.

The unconstrained equations of motion may be obtained by carrying out the steps outlined in Section 2.2 above to yield, for $F_r + F_r^* = 0 \;\; (r =$

$1, 2)$:

$$\begin{aligned} -mg\sin(\phi) - F_f - m\dot{u}_1 = 0 \\ -F_f - \frac{J}{r^2}\dot{u}_2 = 0 \end{aligned} \tag{21}$$

which may be easily solved for $\dot{u}_1$ and $\dot{u}_2$.

The equations of motion for the disk in the rolling phase, with the rolling constraint embedded, may be formulated through a recombination of terms according to (13) and (14). Since there are one constraint and two generalized coordinates, we have $p = n - m = 2 - 1 = 1$, and $D = [-1]$, so Eqs. (13) and (14) yield

$$\begin{aligned} \tilde{F}_1 = F_1 - F_2 \\ \tilde{F}_1^* = F_1^* - F_2^* \end{aligned} \tag{22}$$

which produce

$$-mg\sin(\phi) - \left(m + \frac{J}{r^2}\right)\dot{u}_1 = 0. \tag{23}$$

These operations will actually be undertaken by a symbolic manipulator on-line, during simulation.

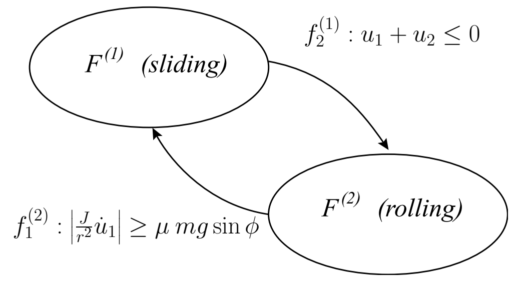
$F^{(1)}$ (sliding)
$f_2^{(1)} : u_1 + u_2 \leq 0$
$F^{(2)}$ (rolling)
$f_1^{(2)} : \left|\frac{J}{r^2}\dot{u}_1\right| \geq \mu\, mg\sin\phi$

Figure 6. Automaton for the Rolling Disk

We are now ready to compose these two sets of equations into a hybrid automaton. Figure 6 shows a hybrid automaton with two modes: mode 1 for the sliding disk and mode 2 for the rolling disk. The switching function $f_2^{(1)}$ that triggers a transition from sliding to rolling is the condition $u_1 = -u_2$ (satisfaction of the rolling constraint). Rolling begins with the initial conditions equal to the final conditions of the sliding mode (i.e., there is no reset function). Rolling ends if ever the force of friction F_f magnitude exceeds the friction cone, defined by $\mu\mathcal{N}$, where $\mathcal{N}$ is the normal force given by $mg\sin\phi$. To produce an algebraic expression for the normal force $\mathcal{N}$ needed to evaluate the tangential friction for the switching function, an auxiliary generalized speed was used to bring $\mathcal{N}$ into evidence.

We have implemented a simulation of this system in MATLAB® using our SymBody toolbox. The pre-processing of symbolic formulation of unconstrained equations of motion takes 1.95 seconds on a 3GHz Pentium IV Processor with 1GB RAM, where online symbolic embedding takes 0.1 seconds and constraint deletion takes 0.03 seconds. We also utilized a fourth order 0.01 second fixed step Runge-Kutta routine with relative tolerance of $1e-7$ for numerical integration. With this routine each step takes about 0.003 seconds in sliding and 0.0025 seconds in rolling mode. We have also implemented a fourth order BDF formula for simulation of formulation using DAEs. Again, for 0.01 second fixed step size and allowing 5 iterations, each step takes about 0.004 seconds in sliding and 0.0045 seconds in rolling mode.

7. Example 2

Example 2 considers a planar system much like the disk B and ramp A of Example 1, except here the ramp is extended by a fence F whose shape is subject to change. This example is inspired by a robotic parts feeding application where parts are sorted by a flexible fence based on real-time computer vision. Another application is a programmable constraint (realized by a robotic materials handling device [28]) whose shape is not known ahead of time but is programmed online to achieve certain goals [24].

To simplify the example, assume the fence F is an arc of length L_f and initially unknown radius R, joined to ramp A at point P, which

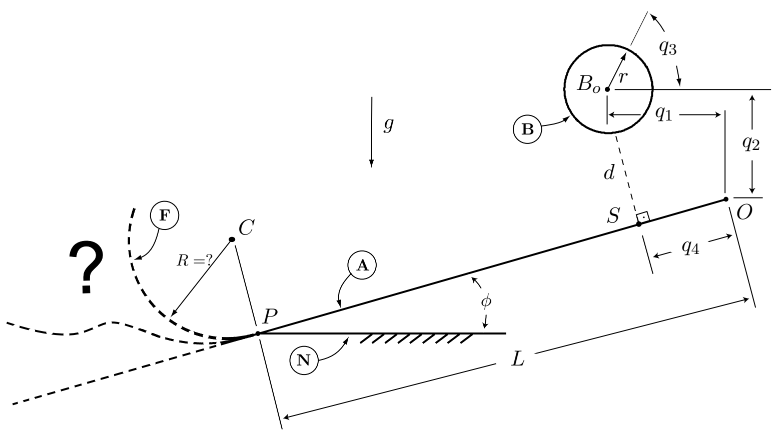

Figure 7. Schematic representation of Example 2

is a distance L from O. Let point C, lying on the perpendicular to A at point P, be the center of F. Let the configuration of B in N be characterized by a set of four generalized coordinates, with q_1, q_2 defined as the horizontal and vertical displacements of B_0 relative to point O, and q_3 defined as the angle between the ramp edge and a line fixed in B that is initially parallel to the ramp edge and q_4 defined as the path length along the ramp (or ramp and fence) that locates the point S along the path that is closest to B. The point S on A is tracked by a feedback stabilized extremal point tracking algorithm discussed in Section 5.2 and the minimum distance between the disk and the ramp is denoted by d.

The automaton for this example, represented in Figure 8, contains three modes: mode 1 for the free disk, mode 2 for the disk constrained to the ramp and mode 3 for the disk constrained to a fence of a certain shape. Since the geometry of the fence is not defined until some time after the simulation is started, the automaton is required to be constructed on the fly and cannot be enumerated beforehand as in Example 1.

The switching functions $f_3^{(2)}$ and $f_2^{(3)}$ that trigger transitions between the ramp and the fence are given by the equation $q_4 = L$.

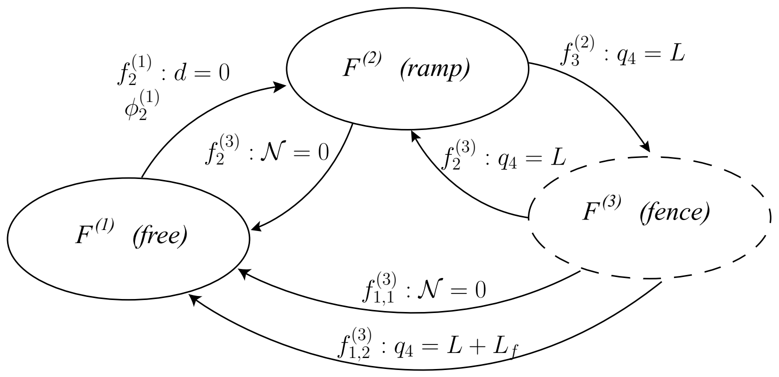

Figure 8. Automaton for the disk-ramp-fence system. The constraint-embedded equations of motion governing the motion of the disk on the fence must be formulated on the fly since the shape of the fence is not known prior to simulation start.

The switching function $f_2^{(1)}$ activates a transition from free mode to a ramp-constrained mode when the disk comes in contact with the ramp. We assume the coefficient of restitution is equal to zero so the reset condition $\phi_2^{(1)}$ sets the velocity of the disk perpendicular to the ramp to zero upon impact. The disk can leave a constraint (the ramp or the fence) when the normal force $\mathcal{N}$ between the disk and the surface becomes tensile, i.e. one of the transition functions $f_1^{(3)}$ or $f_{1,1}^{(2)}$ is satisfied. Finally, the disk can also break free if it reaches the end of the fence, $f_{1,2}^{(2)} : q_4 = L + L_f$. All undefined reset conditions are set to satisfy state continuity.

To embed the constraint equations to obtain the constrained equations of motion, one formulates the configuration constraints defined by the geometry and differentiates them to arrive at motion constraints as explained in Section 2.2. The motion of B in N is defined by generalized speeds: $u_i \triangleq \dot{q}_i$, $(i = 1, \ldots, 4)$. Due to configuration constraints imposed in the ramp and fence modes, the generalized speeds may not all be independent. For example, the motion constraints, obtained by differentiating the configuration constraints due to contact with the ramp, are given as $u_1 = u_4\ \cos(\phi)$ and $u_2 = u_4\ \sin(\phi)$.

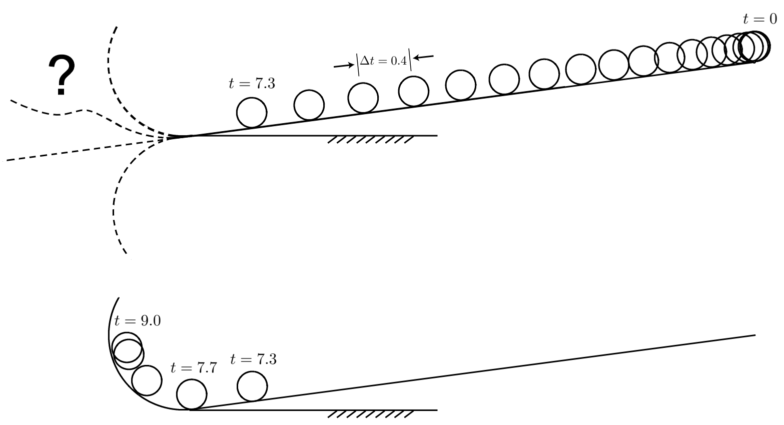

Figure 9. Simulation of a disk constrained to a ramp and a fence whose shape is not predetermined. The top figure shows 19 snapshots taken after the disk comes in contact with the ramp at $\Delta t = 0.4$ sec intervals. The bottom figure shows 5 snapshots taken after the shape of the fence is determined at time $t = 7.3$ sec.

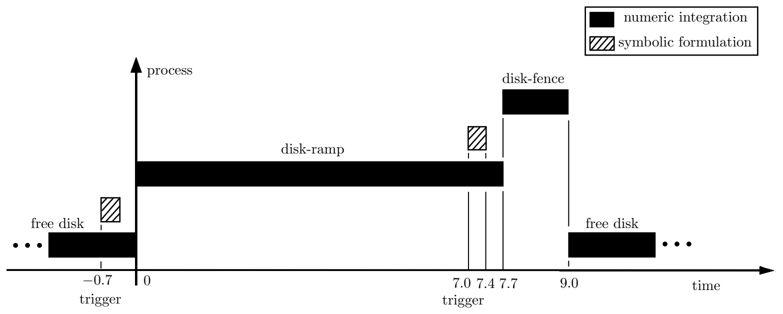

Figure 10. Time chart for Example 2. Symbolic constraint embedding takes place in parallel with numerical integration and is triggered according to a threshold selected such that the embedding is complete before the constrained equations are needed. The time it takes for our toolbox to symbolically embed constraints is 0.4 seconds.

Figure 9 shows 24 snapshots of a simulation at 0.4 second time intervals. The simulation starts at $t = 0$ sec, just after the disk comes in contact with the ramp, with the constraint embedded that ensures disk B maintains contact with ramp A. The shape of the fence is not defined until $t = 7.3$ sec when the sensory feedback becomes available to the robot to program the shape required to achieve the specified task. This circular arc geometry defines a new set of motion (differentiated configuration) constraints: $u_1 = -\frac{(R-r)}{R}\, u_4 \cos(\frac{q_4-L}{R} - \phi)$ and $u_2 = \frac{(R-r)}{R}\, u_4 \sin(\frac{q_4-L}{R} - \phi)$. At time $t = 7.0$ sec constraint embedding for the new mode is triggered according to an arbitrary threshold. During the time interval $t = 7.0$ sec to $t = 7.4$ sec, symbolic constraint embedding for the fence takes place in parallel with numerical integration of the disk-ramp equations of motion. In Figure 10 numerical and symbolic processes running in parallel are shown stacked in the same time interval. Naturally, either a parallel processor or multi-threaded program is required to implement the processes depicted in Figure 10. At $t = 7.7$ sec, the disk comes in contact with the fence and the simulator starts to numerically integrate the new set of disk-fence equations of motion for which the constraint equations ensuring contact with fence F have been embedded. Finally, at $t = 9.0$ sec, the disk reaches the end of the fence and breaks free.

8. Example 3

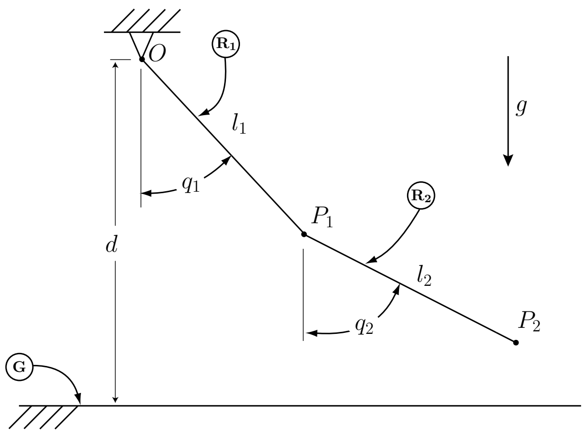

Figure 11. A double pendulum with floor

Figure 11 shows a double pendulum of massless rigid rods R_1 and R_2 of lengths l_1 and l_2 with particles P_1 and P_2 of mass m_1 and m_2

attached. Let generalized coordinates q_1 and q_2 measure the angular displacements of R_1 and R_2 from the vertical. A horizontal floor G is located a distance d below the pendulum pivot O, where $l_1 < d < l_1 + l_2$. Particle P_2 may strike G with a contact model characterized by a coefficient of restitution ϵ that takes on values between 0 and 1 and a coefficient of friction μ.

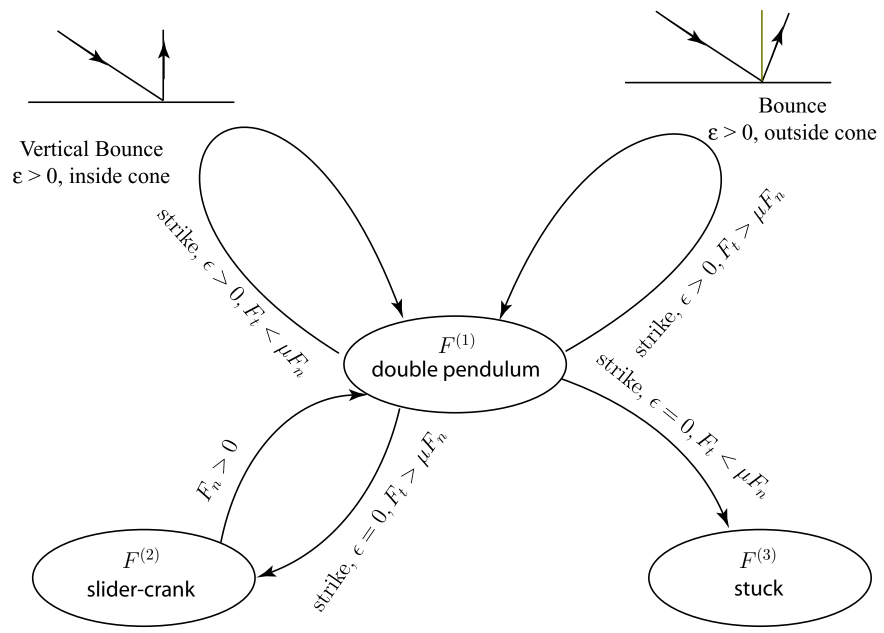
Vertical Bounce
ε > 0, inside cone
Bounce
ε > 0, outside cone
strike, $\epsilon > 0, F_t < \mu F_n$
strike, $\epsilon > 0, F_t > \mu F_n$
$F^{(1)}$
double pendulum
$F_n > 0$
strike, $\epsilon = 0, F_t > \mu F_n$
strike, $\epsilon = 0, F_t < \mu F_n$
$F^{(2)}$
slider-crank
$F^{(3)}$
stuck

Figure 12. Automaton for the Double Pendulum

An automaton depicting the various modes of the double pendulum interacting with the floor G is shown in Figure 12. There are three modes: $F^{(1)}$ representing the unconstrained pendulum with two degrees of freedom, $F^{(2)}$ representing a slider-crank with one degree of freedom, where particle P_2 is constrained to slide along G, and $F^{(3)}$, in which P_2 is stuck on G and the system has no degrees of freedom.

Mode 2 is reachable from mode 1 only when $\epsilon = 0$ and upon striking, the tangential friction force lies outside the friction cone. There is no reset function associated with this transition. Mode 2 transitions back to mode 1 when the normal force becomes tensile: $f_1^{(2)} : F_n > 0$. Mode

3, no motion, is reachable from mode 1 when $\epsilon = 0$ and the friction force F_t lies inside the friction cone.

There are two transitions out and back into mode 1 that invoke reset functions. The first of these, named $f^{(1)}_{1,1}$, is called upon a collision with $\epsilon > 0$ and the friction force lying outside the friction cone, $F_t > \mu F_n$. The second, named $f^{(1)}_{1,2}$, is called upon strike with $\epsilon > 0$ and the friction force lies inside the friction cone, $F_t < \mu F_n$.

Once again, we have implemented a simulation of this system in MATLAB® using our SymBody toolbox. This time, pre-processing of symbolic formulation of unconstrained equations of motion takes 6 seconds on a 3GHz Pentium IV Processor with 1GB RAM, where online symbolic embedding takes 0.3 seconds and constraint deletion takes 0.08 seconds. We also utilized a fourth order 0.01 second fixed step Runge-Kutta routine with relative tolerance of $1e-7$ for numerical integration. With this routine each step takes about 0.008 seconds in slider crank and 0.009 seconds in free mode.

9. Conclusions

In this paper, we presented a simulator designed to handle multibody systems with changing constraints, wherein the equations of motion for each of its constraint configurations are formulated in minimal ODE form with constraints embedded before they are passed to an ODE solver. Unlike other simulator designs, the constraint-embedded equations are formulated symbolically on-the-fly according to a recombination of terms of the unconstrained equations. Constraint embedding undertaken on-the-fly enables the simulation of systems with an ODE solver for which constraints are not known prior to simulation start or for which the enumeration of all constraint conditions would be unwieldy because of their complexity or number. The advantages of this design also include robustness, since issues of drift associated with DAE solvers are sidestepped by symbolic embedding. We also applied nomenclature developed for hybrid dynamical systems to describe the system with changing constraints and to distinguish the roles of the forward dynamics solver, a collision detector, and an impact resolver. Finally, we have prototyped the simulator in MATLAB® and demonstrated the design in three representative examples.

Acknowledgements

This work was supported by a NSF PECASE award CISE-0093290.

References

1. Antsaklis, P., X. Koutsoukos, and J. Zaytoon: 1998, 'On Hybrid Control of Complex Systems: A Survey'. *European Journal of Automation* **32**, 1023–1045.
2. Baraff, D.: 1994, 'Fast contact force computation for nonpenetrating rigid bodies'. In: *SIGGRAPH*. pp. 23–34.
3. Barton, P. I. and C. K. Lee: 2002, 'Modeling, simulation, sensitivity analysis, and optimization of hybrid systems'. *ACM Transactions on Modeling and Computer Simulation* **12**(4), 256–289.
4. Baumgarte, J.: 1972, 'Stabilization of Constraints and Integrals of Motion in Dynamical Systems'. *Computer Methods in Applied Mechanics and Engineering I* pp. 1–16.
5. Bayo, E., J. Garcia de Jalon, and M. Serna: 1988, 'A modified Lagrangian Formulation for the Dynamic Analysis of Constrained Mechanical Systems'. *Computer Methods in Applied Mechanics and Engineering* **71**, 183–195.
6. Branicky, M., V. Borkar, and S. Mitter: 1998, 'Unified framework for hybrid control: model and optimal control theory'. *IEEE Transactions on Automatic Control* **43**(31-45).
7. Brenan, K., S. Campbell, and L. Petzold: 1989, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*. Elsevier.

8. Gear, C., B. Leimkuhler, and G. Gupta: 1985, 'Automatic Integration of Euler-Lagrange Equations with Constraints'. *Journal of Computational and Applied Mathematics* **12**, 77–90.
9. Gillespie, R.: 2003, 'Kanes Equations for Haptic Display of Multibody Systems'. *Haptics-e, The Electronic Journal for Haptics Research* **2**(3).
10. Gillespie, R. B.: 1996, 'Haptic Display of Systems with Changing Kinematic Constraints: The Virtual Piano Action'. Ph.D. thesis, Stanford University.
11. Gilmore, B. and R. Cipra: 1991, 'Simulation of planar dynamic mechanical systems with changing topologies: Part 1–characterization and prediction of the kinematic constraint changes'. *Journal of Mechanical Design* **113**, 70–76.
12. Haug, E., S. Wu, and S. Yang: 1986, 'Dynamics of mechanical systems with coulomb friction, stiction, impact and constraint addition-deletion –I'. *Mechanism and Machine Theory* **21**, 401–406.
13. Huston, R.: 1999, 'Constraint Forces and Undetermied Multipliers in Constrained Multibody Systems'. *Multibody System Dynamics* **3**, 381–389.
14. Javier, G. and E. Bayo: 1993, *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge*. Springer Verlag.
15. Jimenez, P., F. Thomas, and C. Torras: 2001, '3D Collision Detection: A Survey'. *Computers and Graphics* **25**(2), 269–285.
16. Kane, T. and D. A. Levinson: 1999, 'A Multibody Motion Stability Analysis'. *Multibody System Dynamics* **3**, 287–299.
17. Kane, T. R.: 1985, *Dynamics, Theory and Applications*. McGraw-Hill, New York.
18. Keller, J.: 1986, 'Impact with Friction'. *Journal of Applied Mechanics* **583**, 1–4.
19. Kim, S. and M. Vanderploeg: 1986, 'QR Decomposition for State Space Representation of Constrained Mechanical Dynamic Systems'. *ASME Journal on Mechanisms, Transmissions, and Automation in Design* **108**, 176–182.
20. Klisch, T.: 1998, 'Contact Mechanics in Multibody Systems'. *Multibody System Dynamics* **2**, 335–354.
21. Lesser, M.: 1992, 'A Geometrical Interpretation of Kane's Equations'. *Proceedings of the Royal Society of London, Series A.* **436**, 69–87.
22. Lin, M. C. and S. Gottschalk: 1998, 'Collision Detection Between Geometric Models: A Survey'. In: *Proceedings of IMA Conference on Mathematics of Surfaces*. pp. 602–608.

23. Lötstedt, P.: 1982, 'Mechanical systems of rigid bodies subject to unilateral constraints'. *SIAM Journal of Applied Math.* **42**(2), 281–296.

24. Lynch, K. M., C. Liu, A. Sorensen, S. Kim, M. Peshkin, J. E. Colgate, T. Tickel, D. Hannon, and K. Shiels: 2002, 'Motion Guides for Assisted Manipulation'. *International Journal of Robotics Research* **21**(1), 27–43.

25. Mitiguy, P.: 1995, 'Efficient formulation and solution of equations of motion'. Ph.D. thesis, Stanford University.

26. Orlandea, N., D. Calahan, and M. Chace: 1977, 'A Sparsity-oriented Approach to the Dynamic Analysis and Design of Mechanical Systems: Part I and Part II'. *Journal of Engineering for Industry* **3**(99), 773–784.

27. Patoglu, V. and R. B. Gillespie: 2002, 'Extremal Distance Maintenance for Parametric Curves and Surfaces'. *ICRA 2002* **3**, 2717–2723.

28. Peshkin, M. A., J. E. Colgate, W. Wannasuphoprasit, C. A. Moore, R. B. Gillespie, and P. Akella: 2001, 'Cobot Architecture'. *IEEE Transactions on Robotics and Automation* **17**(4), 377–390.

29. Reckdahl, K. J.: 1997, 'Dynamics and control of mechanical systems containing closed kinematic chains'. Ph.D. thesis, Stanford University.

30. Riley, S. M.: 2000, 'Model Reduction of Multibody Systems by the Removal of Generalized Forces of Inertia'. Ph.D. thesis, The University of Michigan, Ann Arbor.

31. Sayers, M. W.: 1990, 'Symbolic Computer Methods To Automaically Formulate Vehicle Simulation Codes'. Ph.D. thesis, The University of Michigan, Ann Arbor.

32. Schaechter, D. B. and D. A. Levinson: 1988, 'Interactive computerized symbolic dynamics for the dynamicist'. *Journal of Astronautical Sciences* **36**(4), 365–388.

33. Serna, M., R. Aviles, and J. Garcia de Jalon: 1982, 'Dynamic Analysis of Plane Mechanisms with Lower-Pairs in Basic Coordinates'. *Mechanism and Machine Theory* **17**, 397–403.

34. Singh, R. and P. Likins: 1985, 'Singular Value Decomposition for Constrained Dynamic Systems'. *ASME Journal of Applied Mechanics* **52**, 943–948.

35. Smith, C. E.: 1991, 'Predicting Rebounds Using Rigid Body Dynamics'. *Journal of Applied Mechanics* **58**, 754–758.

36. Wampler, C., K. Buffington, and J. Shu-hui: 1985, 'Formulation of equations of motion for systems subject to constraints'. *Journal of Applied Mechanics* **52**, 465–470.

37. Wang, Y. and M. T. Mason: 1992, 'Two-dimensional rigid-body collisions with friction'. *Journal of Applied Mechanics* **59**, 635–642.

38. Wehage, R. A. and E. J. Haug: 1982, 'Dynamic analysis of mechanical systems with intermittent motion'. *Transactions of the ASME* **104**, 778–784.