Simbody™

(The SimTK Multibody Dynamics Toolset)

Requirements for SimTK 1.0

Michael Sherman

Version 1.0, August 19, 2005

Abstract

Here we discuss the requirements for a SimTK core capability for multibody dynamics (a.k.a. internal/torsion/relative coordinate modeling). This toolset, to be called "Simbody," is intended to be useful in coarse grain molecule modeling, neuromuscular gait simulation, and many other biologically relevant models. We provide some background here and then narrow that to specific requirements for Simbody 1.0 to be included as part of SimTK 1.0.

1 Purpose of this document 1
2 Background 2
  2.1 What is "multibody dynamics"? 2
    2.1.1 Components of a multibody model 3
    2.1.2 A comment on deformable (flexible) bodies 3
  2.2 What does multibody dynamics do for Simbios? 3
3 What do we need in Simbody 1.0? 4
4 What are we leaving out in 1.0? 5
5 Implementation strategy 5
  5.1 Layered design 6
  5.2 C++, C, Fortran, and CCA interfaces (APIs) 7
  5.3 Start with existing code 7
  5.4 Structure to work with SimTK Modeling Layer 7
6 Testing and validation 7
7 Summary 7
8 Appendix: Basic multibody theory 8
  8.1 Preliminaries 8
    8.1.1 Bodies 8
    8.1.2 Joints 8
    8.1.3 Forces 9
    8.1.4 Constraints 9
  8.2 Kinematics 9
  8.3 Dynamics 10
  8.4 Equations of motion 10
    8.4.1 Unconstrained systems 10
    8.4.2 Constrained systems 11
  8.5 Modal analysis and implicit integration 12
  8.6 Root finding and optimization 13
Acknowledgments 13
References 13

1 Purpose of this document

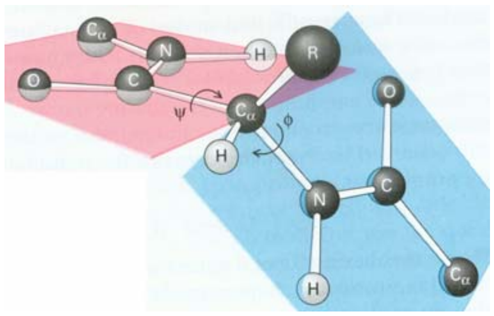
This document provides background describing the Simbios needs addressed by multibody dynamics, introduces the technology, and the SimTK Multibody Dynamics Toolset which we call "Simbody." It is intended as a way to communicate and come to a consensus regarding our plans in this area for SimTK 1.0, and to guide the development of software which implements these plans. This is not intended, however, as a detailed software specification or user's manual.

2 Background

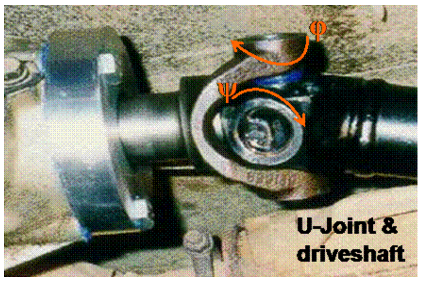
This is general material hopefully providing enough background for the rest of the document to make sense. Even for those familiar with multibody dynamics, it is probably worth reading to see how we are characterizing it for the broad uses it will serve for Simbios.

2.1 What is "multibody dynamics"?

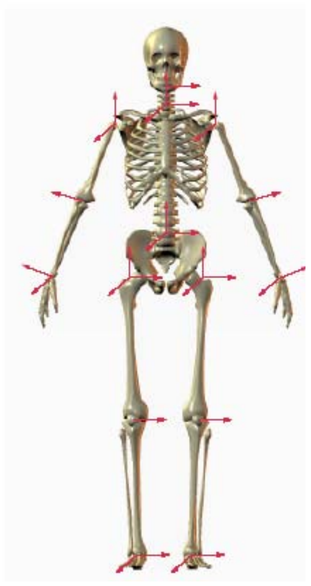
Multibody dynamics is the field studying the classical mechanical properties (especially motion) of systems of *bodies* interconnected by *joints*, influenced by *forces*, and restricted by *constraints*. The key feature of a system that makes it suitable for multibody treatment is the observation that the motion is *localized*, that is, it is well-described as a set of composite parts which undergo large motion with respect to one another, but are themselves nearly rigid. Figure 1 shows some examples of the breadth of applicability of multibody dynamics, which has been used effectively to model machines, skeletal motion and gait, coarse-grained biopolymers, and many other systems relevant to a wide variety of scientific and engineering disciplines.



Protein backbone



Mechanical U-joint



Articulated skeleton

Figure 1: Some multibody systems.

Multibody dynamics is a generalization of several more-familiar modeling methods. It includes as special cases, for example, systems of point masses represented in Cartesian coordinates (e.g. molecular dynamics models) and systems of freely moving extended bodies (typically, rigid bodies). Multibody dynamics should be viewed as a basic numerical capability fundamental to any simulation system. It is in the same category as, say, a linear algebra library, not an end-user application. Simbody will be used by modelers and application developers as a basic building block. Simbios computational researchers (algorithm inventors) can use Simbody as a baseline source of correct answers for debugging and as a point of comparison to demonstrate the superiority of their new methods.

2.1.1 Components of a multibody model

All mass and geometric features of the system are associated with the bodies (with "Ground" viewed as an immobile body). Large scale motion is permitted only at joints, whose degrees of freedom (dofs) define *generalized coordinates* describing the system configuration in terms of relative translations and orientations of the bodies they interconnect, and *generalized speeds* describing the relative motion of those bodies. Generalized coordinates are sometimes referred to as "internal coordinates," "relative coordinates," or "torsion coordinates."

Forces (more properly *generalized forces*) include both forces and moments (torques) and may be applied to bodies or directly along a joint coordinate. Constraints express algebraic restrictions on the allowed values of the generalized coordinates and speeds. One may reasonably think of constraints as "infinitely strong" forces.

As a practical matter, we consider bodies and joints to be the fundamental features of a multibody system, together defining the system's *topology* which is invariant. A change in the number of bodies, connectivity of joints, or joint types results in a new multibody system. Forces and constraints, on the other hand, can be added, changed, and removed from a multibody system without changing its identity. This does not imply that topology must remain fixed during an investigation, just that a topology change is a more significant operation than a change in forces or constraints.

2.1.2 A comment on deformable (flexible) bodies

In general, the bodies of a multibody system do not have to be rigid. It is sometimes desirable to allow the bodies themselves to undergo small internal motions, called *deformations*. These add a new set of independent coordinates to the overall system coordinates and speeds, but we distinguish them from the generalized coordinates and generalized speeds introduced by joints and refer to them instead as *deformation coordinates* and *deformation rates*. Various techniques can be used to determine the appropriate representation of deformable bodies. Such bodies can be used, for example, to supply "ring pucker" coordinates for molecules rather than modeling the mobility of every bond individually. Or, the techniques of structural mechanics can be used to aggregate large nearly-rigid subsystems into deformable bodies with "assumed mode" linear deformations.

We will not support deformable bodies in SimTK 1.0, but will allow for adding them in the future (e.g. by not building in an assumption that a body's center of mass is in a fixed location in the body frame). In the meanwhile it is always possible to model body flexibility by partitioning the body into joint-connected rigid bodies, with internal forces and constraints modeling the deformation behavior.

In current practice, systems of rigid bodies have been extremely useful in many fields while systems of interconnected deformable bodies, while intriguing and occasionally useful, have not yet found wide application.

2.2 What does multibody dynamics do for Simbios?

Multibody dynamics is a necessary ingredient for three of the four current Simbios Driving Biological Problems (DBPs). It is already used routinely in musculoskeletal modeling for investigation of biomechanics of gait and in design of artificial joints and prosthetics. Scott Delp's lab makes extensive use of multibody dynamics, using a commercial package (SD/FAST). Multibody dynamics is also being used in modeling the myosin/actin molecular motor in research being done by David Parker in Jim Spudich's lab, again using SD/FAST and more recently the SCI package from Oussama Khatib's lab (which is also proprietary). The RNA structure DBP uses coarse grained rigid bodies to represent RNA. In addition, both Vijay Pande and

Michael Levitt perform extensive molecular mechanics work using multibody dynamics codes which are specialized to molecule simulation (usually in the form of Cartesian point-mass models), and Michael Levitt pioneered the use of internal coordinates for molecule modeling in the 1970's.

SimTK needs to provide an open-source alternative to SD/FAST that will make this capability available to a wider audience and also remove some of the limitations imposed by SD/FAST's 20-year-old technology. Simbody can serve as a direct replacement for SD/FAST, and we should provide a compatible interface. Properly designed, a general multibody capability can also be plugged in to existing molecular simulation frameworks to permit construction of coarse-grained molecular models using conventional force fields. That is a more substantial effort than a plug-compatible replacement, however, because today's popular molecular dynamics codes do not support internal-coordinate dynamics. Simbody 1.0 should be structured to facilitate this incorporation.

3 What do we need in Simbody 1.0?

Basic capabilities (all operations are fast, O(n) operations unless otherwise stated):

- Given a set of atomic positions for a polymer, and an internal coordinate (multibody) model, find the set of internal coordinates that best represents the given atomic positions.
- Given a multibody model and values for its generalized (relative) coordinates and speeds, provide the corresponding spatial locations and velocities (kinematics).
- Given values for generalized coordinates, speeds and accelerations, calculate the internal (joint) forces which would have produced those accelerations (inverse dynamics).
- Given a set of spatial forces, calculate the equivalent system of forces which act only at the generalized coordinates.
- Calculate accelerations in internal coordinates. That is, calculate instantaneous generalized speed time derivatives. (forward dynamics)
- Support both general model building (specify bodies and joints) and specialized modeling for constructing protein and nucleic acid multibody models from, e.g., pdb files.
- Calculate matrices and operators needed for Operational Space Control (kinematic Jacobian, partial velocities).
- Calculate matrices needed for calculating normal modes in internal coordinates (also needed for implicit integration). (dynamic Jacobian) This is a relatively expensive operation – inherently O(n^2) but may be even worse in the 1.0 release.
- Provide interfaces to Simbody which are callable from C++, C, and Fortran as well as a prototype CCA "Port" interface.
- Stateless design for compatibility with modeling layer.
- Provide an SD/FAST-compatible interface, both for ease of conversion and to facilitate building tests which compare Simbody results and performance with the same problem solved with SD/FAST. This should include most SD/FAST functionality including joint types, application of forces, and addition of constraints.
- Provide (or at least suggest) numerical methods, not strictly a part of the Simbody toolset, which are useful for manipulating multibody models. This includes time integration, modal analysis, root finding, initial condition analysis, and solving impulse problems.
- Provide suitable documentation enabling users to use Simbody effectively. Note that all users are programmers since Simbody is an API (callable library) rather than an application.

4 What are we leaving out in 1.0?

- All bodies will be rigid. Deformability, if desired, will be achieved by using multiple rigid bodies interconnected with appropriate joints, constraints, and forces.
- Matrices needed for implicit integration and normal modes may be calculated by numerical differencing in 1.0, with analytic methods to follow later.
- Model building capability will be limited, in the sense that convenient, domain-specific modeling building facilities will not be ready. Fully general, "lowest common denominator" model building will be available, but will require a greater level of multibody knowledge that is ultimately necessary. As an example, an automated protein-to-torsion-angle model mapping a pdb file directly to a multibody model is feasible and useful, but probably will not be done by SimTK 1.0. Instead, early application writers may have to parse their own molecules, decide how to split them into bodies, and then call the basic multibody modeling tools to create the model.
- Constrained systems will use a method which involves factoring a potentially singular matrix whose size is the number of constraints. This yields an extremely robust solution, but would be a performance problem in highly constrained systems. This is likely insignificant in most biological systems but will come up sometimes and improvements are possible.
- Integration with existing applications (e.g. Gromacs) will be absent or incomplete.
- Performance will be suboptimal at first release.
- Wrappers for interpreted languages like Java and Python will not be available in SimTK 1.0. There will be C, Fortran, and C++ access only. We do intend to support these later.
- SD/FAST emulation will not be perfect, but should be close enough to make transition straightforward.

5 Implementation strategy

To satisfy the needs of varied Simbios users, Simbody should be implemented in several layers, as shown in Figure 2.

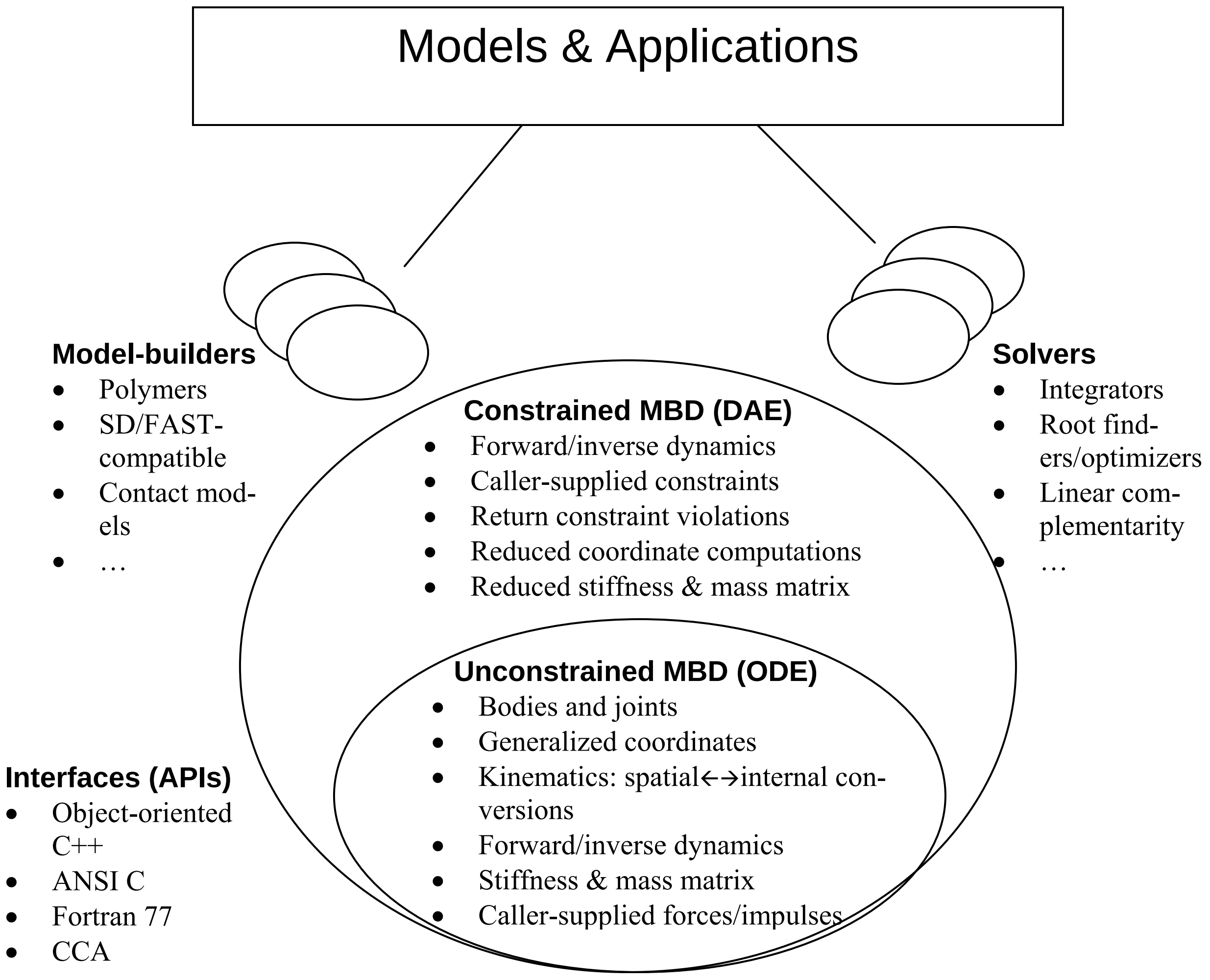


Figure 2: Layered architecture of the SimTK Multibody Dynamics Toolset (Simbody)

5.1 Layered design

The lowest level is a library capable of addressing unconstrained ("tree") systems as defined by Equation 1 in the Appendix. This level should understand nothing but body and joint definitions, and provide the ability to map generalized coordinates and speeds into spatial locations and velocities (kinematics), and to map spatial forces into joint forces. It should provide operators for calculating the accelerations given user-supplied forces. The resulting dynamic system is an ODE.

The next layer adds constraints to the above. It will make use of the tree layer to solve the constrained problem as expressed in Equation 4 in the Appendix. It provides for caller-supplied constraints in addition to the bottom layer's caller-supplied forces. The result is an overdetermined DAE, in which the accelerations automatically satisfy the constraints, but where the velocity and position constraints will not be satisfied. This layer can calculate the constraint errors and return them.

The next layer out is for creating models. This will include at least an SD/FAST-compatible model builder which can read SD/FAST input files, and a molecule modeler which can construct rigid bodies from grouped atoms. In many cases modeling is a multi-step process in which Simbody is used to analyze "base" models to aid in constructing "lumped" models of those same systems.

Related to, but possibly outside of the Simbody toolset, is a collection of solvers. These are generally independent, generic numerical methods, but in some cases multibody models are designed to work with particular solvers.

5.2 C++, C, Fortran, and CCA interfaces (APIs)

Orthogonal to the above are the interfaces available to programmers (APIs). For each layer, we will provide an object-oriented API for use by C++ programmers, and procedural interfaces callable from C and Fortran programs.

In addition, if there is time we will provide a prototype of a Common Component Architecture interface to all or part of Simbody. This entails defining an appropriate Port or Ports, and writing a CCA neoclassic wrapper in C++ for the Simbody libraries.

5.3 Start with existing code

We will build the new SimTK Multibody Toolset on a foundation of well-written, well-tested, pre-existing, open source code. We have identified two reasonable possibilities at this point: the IVM module[1] written by Charles Schwieters at the NIH, and the TAO package owned by Arachi and based on software originally developed in Oussama Khatib's lab at Stanford.[2] The latter is not currently open source, but there is some possibility that its owners will remove the current restrictions so that it can be incorporated in SimTK. We do have the last open source version of this package (called PrRobot) in SimTK, but it does not have sufficient capability to serve as a good base for the SimTK Multibody Toolset.

In addition, Kurt Anderson at Rensselaer Polytechnic has proposed his new "POEMS" multibody system as a core tool for SimTK. This is likely to be a very good implementation, however it is not far enough along yet to serve as our first "reference" implementation. It could be a great addition or replacement later, however.

5.4 Structure to work with SimTK Modeling Layer

Although Simbody is a computational library rather than a Subsystem, mechanical Subsystems can easily be built using it. We should build it following SimTK modeling guidelines, most importantly to ensure that the Simbody libraries are stateless and perform evaluation in identifiable stages. This will also serve to give us some experience with restructuring existing code to make it stateless. The experience gained can then be passed on to other SimTK developers who may need to follow the same path.

6 Testing and validation

Compare with PrRobot, SD/FAST and TAO if we can get it. If possible, preserve the existing IVM test cases. Tests should cover both functionality and problem size, so that we can track performance improvements.

Tests should include a variety of simple models with easily verifiable analytical answers, as well as complex models constructed independently and compared.

7 Summary

For SimTK 1.0, we need a basic multibody capability which provides a robust replacement for existing SD/FAST use, as well as the additional flexibility needed for effective modeling of large molecular systems. This must be accessible to users who work at different levels, from Fortran and C to object-oriented C++. The structure of the resulting system should be compatible with our proposed modeling approach (e.g., stateless) and should serve as an example of the approach we are taking to numerical libraries in general, designed with standardized interfaces defining interchangeable numerical methods.

Simbody 1.0 will address these issues.

8 Appendix: Basic multibody theory

Some readers may find this more-technical discussion helpful in defining the specific approach we have in mind; others will find it confusing and perhaps somewhat irrelevant and are invited to skip it!

8.1 Preliminaries

Here are some elaborations on the fundamental objects of a multibody system.

8.1.1 Bodies

Fundamentally, a body B is a moving reference frame (called the body frame B), consisting of a reference point $\mathcal{O}^B$ and three orthogonal reference directions (unit vectors) $\mathbf{x}^B$, $\mathbf{y}^B$, $\mathbf{z}^B$.[*] A distinguished body "Ground" represents the inertial (fixed) reference frame G, providing a global origin $\mathcal{O}$ and fixed directions $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$. The measure numbers (numerical values) of the reference directions can be arranged as the columns of a 3×3 orthogonal rotation matrix $\mathbf{B}$=[$\mathbf{x}^B$ $\mathbf{y}^B$ $\mathbf{z}^B$] or $\mathbf{G}$=[$\mathbf{x}$ $\mathbf{y}$ $\mathbf{z}$]. Thus reference frame B={$\mathbf{B}$, $\mathcal{O}^B$} and G={$\mathbf{G}$, $\mathcal{O}$}. The *i*'th body is B_i, and by convention $B_0 \equiv G$.

Bodies typically have associated properties expressed in the body frame. These include vectors and *stations*, which are point locations, as well as more general geometry. The body frame origin is the station whose measure numbers when expressed in the body frame are [0,0,0]. Mass properties include the total mass (a scalar), the center of mass (a station), and an inertia tensor (3×3 symmetric matrix) which expresses rotational inertia about a particular station. When the inertia tensor is defined about the center of mass it is called the centroidal inertia.

Singular bodies are those whose centroidal inertia has a zero diagonal element. Bodies consisting of a single point mass (such as an atom) have a centroidal inertia of exactly zero. Bodies consisting of only collinear point masses (such as two bonded atoms) have a single zero inertia diagonal, meaning that they have no inertia about their common axis. Bodies with three or more atoms, not all collinear, have non-singular inertia matrices. Massless bodies are also singular.

Deformable bodies, not included in 1.0, are represented the same way but introduce deformations about the body frame. This causes stations and directions which are constant for rigid bodies (e.g., center of mass) to have state dependence for flexible bodies. However, the deformations are always expressed with respect to a rigid body frame associated with the body. That is, every deformable body is built on an underlying rigid body in whose frame its deformations are expressed.

8.1.2 Joints

Joints connect a pair of distinct bodies (and remember that Ground is a body) and define the relative motion (degrees of freedom or "dofs") allowed between those bodies. The parameterization of these dofs is a set of generalized coordinates q representing the joint configuration, and generalized speeds u representing the joint motion.

The three fundamental joint types are sliding, torsion, and orientation. A *sliding joint* (syn: prismatic joint) provides a single degree of freedom representing translation along a defined axis, and adds a single coordinate with units of length to the system's set of generalized coordinates. A *torsional joint* (syn: pin joint) provides a single degree of freedom representing rotation about a defined axis and adds a single generalized coordinate with angular units. An *orientation joint* (syn: ball joint, spherical joint) permits unrestricted relative orientation between its pair of bodies, that is, three degrees of freedom and at least three corresponding generalized coordinates (for dynamics these require a four-element quaternion).

[*] We will always follow a right-handed convention so that $\mathbf{z}^B = \mathbf{x}^B \times \mathbf{y}^B$.

All other joint types can be viewed as compositions of the three basic types. For example, a *Cartesian joint* is a composition of three sliding joints with orthogonal axes and thus permits unrestricted relative translation (three degrees of freedom) between its bodies. A *free joint* is a composition of a Cartesian joint and an orientation joint and permits six degrees of freedom (completely unrestricted motion) between its bodies. A free joint serves to introduce free bodies into the system and simply provides a convenient reference frame and corresponding coordinates with which to express their motion. Note that, like any other joint, a free joint can be placed between any two bodies—it does not have to connect a body to ground. This allows very convenient relative coordinates to be used for collections of independent bodies. For example, one can express a protein domain that carries its local waters and ions along with it when it is moved kinematically.

More complex joints can be built up from joints and constraints. A "screw joint" for example can be composed of a coaxial sliding and torsion joint, providing one translational and one rotational coordinate, plus a (holonomic) constraint enforcing a defined relationship (the screw's "pitch") between the time derivatives of these coordinates.

[Note: explore whether complex joints can be created directly as joints via a more elaborate joint transition matrix. This could permit a single degree of freedom joint that behaved like a knee, for example.]

8.1.3 Forces

By forces we mean "generalized forces" which includes both forces and torques (moments). Force vectors can be applied to the multibody system at any body station and moment vectors can be applied to any body (or implemented as pairs of forces). Scalar forces or torques can also be applied directly to joint axes, that is, directly along the generalized coordinates. All systems of forces can be reduced to an equivalent set acting only on at the joint coordinates, and Simbody will provide an operation which performs this conversion.

Forces can be functions of time, configuration, or velocity. They may be local effects or result from spatially distributed fields or a constant gravitational field, or act pairwise between distant stations (e.g. atoms) in the system. Forces which depend only on time and configuration are called conservative forces, and are the gradient of some potential function. Non-conservative forces may depend on velocities as well.

8.1.4 Constraints

Constraints may represent arbitrary restrictions on the generalized coordinates and generalized speeds, and linear restrictions on accelerations. Constraints arise, for example, if the body/joint connectivity graph contains a loop. Each independent constraint removes one degree of freedom from the system. In this sense constraints are the complement of joints, whose generalized speeds each *add* one degree of freedom to the system. And in fact any n-dof joint can be represented instead as a free joint plus $6–n$ constraints.

Constraints among the moving bodies of a physical system act by introducing non-working internal forces and moments. These forces act exactly as do the applied forces described above—they can act on bodies or along joint axes, and as with applied forces they can always be reduced to a system of forces acting along the joint axes.

8.2 Kinematics

This refers to the mapping between generalized coordinates and speeds and their spatial counterparts. For example, given values for the generalized coordinates, one should be able to obtain (cheaply) positions and orientations for bodies and spatial (Cartesian) locations of any stations (e.g. atoms). In the other direction, one should be able to apply spatial forces and calculate the equivalent set of joint axis forces that would produce the same motion. Energy calculations can thus be performed using only kinematics.

Kinematic results available in SimTK 1.0 should be sufficient to permit the solution of kinematic problems such as finding the set of generalized coordinates which best approximates a given set of spatial locations. Such problems arise, for example, when fitting a reduced-coordinate molecular model to a set of atom positions determined with X-ray crystallography. More generally, there is a broad assortment of useful initial condition analyses which must be performed prior to the start of a dynamic analysis, and these are based on kinematic calculations.

8.3 Dynamics

Dynamics refers to the relationship between forces and motion. There are two flavors: forward dynamics, in which forces are known and motion calculated, and inverse dynamics where motion is known and forces are to be calculated. Various combinations of known and unknown forces and motions are possible. Simbody should support both of these operations and provide access to the basic O(n) operators that manipulate the associated quantities.

Note that Simbody itself focuses on instantaneous dynamics, that is, the relationship between forces and accelerations at a particular time and state. This capability is designed to be used in conjunction with numerical methods, primarily numerical integrators, to advance the time and state. These numerical methods exist independently of Simbody, however we will coordinate efforts to make sure that suitable methods are available and work smoothly together.

8.4 Equations of motion

Given the above description, we can write down the system of equations defining a multibody system. These are written in terms of the n_q generalized coordinates q and n_u generalized speeds u, which arise from the presence of joints. Generalized speeds are more fundamental than generalized coordinates. The number of degrees of freedom n introduced by the joints is always $n=n_u$, while $n_q \geq n$ because of quaternions. Note that n is the *unconstrained* system degrees of freedom, the net dofs after constraints will be $n–m$ where m is the number of independent constraints.

It should be emphasized that this is a formal description, and that it would be extremely inefficient to set up and solve the equations in the form they are presented below. The techniques of multibody dynamics provide the solution of these equations without ever requiring their explicit formation.

A few conventions: We use n and subscripted n's to count quantities related to coordinates (degrees of freedom) and m and subscripted m's to count constraints. We use overdot to represent differentiation with respect to time.

8.4.1 Unconstrained systems

In a system with no constraints, the equations of motion are

$$\dot{q} = \mathbf{Q}(q)u \tag{1a}$$

$$\mathbf{M}(q)\dot{u} = \mathbf{f}(t,q,u) \tag{1b}$$

Here **M** is an $n\times n$, symmetric, positive definite mass matrix which captures all the inertial properties of the system in its current configuration, and **f** is the set of n forces and torques acting along the joint axes which is equivalent to all the forces and moments applied to the system along with coriolis and gyroscopic terms and gravity. **Q** is an $n_q\times n$ ($=n_q\times n_u$) invertible mapping between generalized speeds and generalized coordinate derivatives (in practice this is used to convert angular velocities to quaternion derivatives). Note that equation (1b) is just Newton's second law F=ma written backwards, as is the strange custom among multibody dynamicists!

Formally, we can solve for the accelerations $\dot{u}$ with

$$\dot{u} = \mathbf{M}^{-1}\mathbf{f} \tag{2}$$

By formally we mean, “don’t take this literally”! There is always special structure to **M** that can be exploited such that the accelerations can be calculated directly in O(n) time, while a literal matrix inversion would take O(n^3) time and be prohibitive for large systems.

As an example, consider the special case of a molecular system modeled with n_a point mass atoms and Cartesian coordinates. **M** is then a diagonal matrix of dimension $3n_a \times 3n_a$ with the atomic masses (each repeated three times) arrayed along the diagonal. The q’s are the Cartesian coordinates, and the u’s are the Cartesian velocities so $n_q=n_u$, **Q** is an identity matrix, and $\dot{q} = u$. **f** is simply the Cartesian forces acting on each coordinate of each atom, typically resulting from taking the gradient of the potential energy function. This represents a set of $3n_a$ uncoupled scalar equations for the Cartesian accelerations of each atom.

In a more general multibody system **M** will be dense as a result of coupling produced by the internal coordinates. Use of quaternions for orientation results in there being more q’s than u’s and **Q** is no longer identity. However, equation (2) provides the solution for the accelerations in this case just as well, and the special structure of multibody systems permits a solution in O(n) time regardless of the amount of coupling in **M**.

8.4.2 Constrained systems

Constraints introduce unknown, non-working forces and torques into the system. Constraints are introduced, for example, if there are topological loops created by the set of bodies and joints. The constraint forces are additional unknowns (along with accelerations). We call these unknowns Lagrange multipliers and represent them as a vector λ. These are mapped to joint forces with a coupling matrix **A** and thus modify acceleration Equation 1b like this:

$$\mathbf{M}\dot{u} = \mathbf{f} - \mathbf{A}^{\mathrm{T}}\lambda \tag{3a}$$

$$\mathbf{A}\dot{u} = \mathbf{b} \tag{3b}$$

where $\mathbf{A}_{m\times n}=\mathbf{A}(t,q,u)$ and $\mathbf{b}_{m\times 1}=\mathbf{b}(t,q,u)$, m is the number of constraints and $n=n_u$ is the number of generalized speeds. Equation 3 has $n+m$ equations in $n+m$ unknowns so can be solved for the accelerations that satisfy the constraint equations.[*] The solution of this system makes use of the unconstrained result from Equation 2. Note that because we can directly solve for $\dot{u}$ and eliminate λ, this is still just an ordinary differential equation, with $\dot{u} = \dot{u}(t,q,u)$.[†]

Equation 3 was written in terms of linear constraints on the accelerations $\dot{u}$. However, in most cases constraints are known only at the configuration level, that is, as (nonlinear) algebraic relationships which must hold among the q’s or among quantities fully determined by the q’s. A constraint like “these two atoms must be a certain distance apart at all times” would be an example. In other cases the constraints may be expressed at the velocity level as restrictions on u. In these cases we differentiate the constraints once or twice until we have corresponding acceleration constraints, and then use them in Equation 3.

Following this procedure yields correct accelerations, but with approximate numerical integration of those accelerations the original position or velocity constraints will not remain satisfied over time. In practice, any constraints that are not actively enforced will gradually drift apart during a dynamic simulation. To address this, we must keep the original algebraic constraints in the problem and solve them along with the ODE in Equation 3. That results in a system of mixed differential and algebraic equations, known as a DAE. The complete system then looks like this:

---

[*] In general the constraint matrix **A** can be singular, so there may be no solution, or an unlimited number of solutions, in which case least squares solutions for λ are typically used.

[†] Knocking Equation 3 around a little, one can verify that $\dot{u} = \dot{u}_0 + \dot{u}_C$, where $\dot{u}_0 = \mathbf{M}^{-1}\mathbf{f}$, $\dot{u}_C = -\mathbf{M}^{-1}\mathbf{A}^{\mathrm{T}}\lambda$, and $\lambda = (\mathbf{A}\mathbf{M}^{-1}\mathbf{A}^{\mathrm{T}})^{+}(\mathbf{A}\dot{u}_0 - \mathbf{b})$, with superscript “+” the conventional notation for pseudoinverse. Again, this is a *formal* solution—it would be prohibitively expensive to solve the way it is written here.

$$\dot{q} = \mathbf{Q}u \tag{4a}$$

$$\mathbf{M}\dot{u} = \mathbf{f} - \mathbf{A}^{\mathrm{T}}\lambda \tag{4b}$$

$$\mathbf{A}\dot{u} = \mathbf{b} \tag{4c}$$

$$\mathbf{V}u = \mathbf{c} \tag{4d}$$

$$\mathbf{g}(t,q) = 0 \tag{4e}$$

where $\mathbf{V}$=$\mathbf{V}(t,q)$ is an $m_{\mathrm{v}}\times n$ matrix, $\mathbf{c}$=$\mathbf{c}(t,q)$ is $m_{\mathrm{v}}\times 1$, and $\mathbf{g}$ is $m_{\mathrm{p}}\times 1$ nonlinear constraints involving t and q. (We're assuming only linear constraints on velocities here, although that is not strictly necessary.) This is an overdetermined system since there are more equations than unknowns. Equations 4abc is an ODE that can be solved for coordinate derivatives $\dot{q},\dot{u}$ which can be integrated using any suitable ODE method to produce a dynamic trajectory for q and u. Equations 4de are used at each step to evaluate how well the resulting trajectory satisfies the constraints, and a wealth of methods that have been developed to solve overdetermined DAEs can then be applied to ensure constraint satisfaction.[3] For Simbody 1.0, we should support at least the technique known as *coordinate projection*, which is very accurate and reliable in practice.[4]

For those familiar with molecular simulation, the SHAKE procedure is an example of an approach to enforcing a position constraint such as that represented in 4e. SHAKE performs a projection guaranteeing that a molecular trajectory will preserve certain known bond lengths.

It is worth mentioning that in the common case where the only constraints are provided at the position level via equation 4e, equation 4d comes from differentiating 4e, and 4c from further differentiation of 4d. In that case (ignoring quaternions) we have $\mathbf{A} = \mathbf{V} = \mathbf{G} = \partial\mathbf{g}/\partial q$.

8.5 Modal analysis and implicit integration

In this section we discuss the related needs of modal analysis (that is, normal modes in internal coordinates) and implicit integration. Both of these require that the system equations of motion be differentiated with respect to the generalized coordinates and speeds. That is we want to calculate the dynamic, internal coordinate Jacobian

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{qq} & \mathbf{J}_{qu} \\ \mathbf{J}_{uq} & \mathbf{J}_{uu} \end{bmatrix} = \begin{bmatrix} \partial\dot{q}/\partial q & \partial\dot{q}/\partial u \\ \partial\dot{u}/\partial q & \partial\dot{u}/\partial u \end{bmatrix} \tag{5}$$

Modal analysis is typically done with all speeds set to zero, so only the submatrix $\mathbf{J}_{uq}$ is of interest. If q is such that the system is stable (at a local energy minimum), then the eigenvalues of this matrix are the normal modes of the system about that equilibrium point and the corresponding eigenvectors are the modal basis (that is, they represent the coordinated motion involved in each of the normal modes).

Given the system equations of motion, note that one can easily obtain an approximation to **J** by perturbing the state variables (this is called a finite difference approximation to **J**). Simbody 1.0 should, at a minimum, support that method. However, it is both inaccurate and extremely expensive to compute. Finite differencing loses about half the available precision, and requires O(n) calculations of the system accelerations to form an $n\times n$ matrix. In molecular dynamics straightforward force calculations are typically O(n^2), so this can mean the Jacobian calculation is a prohibitive O(n^3). In any case the force calculations are very expensive and doing O(n) of them to get a half-accurate Jacobian is not a very good deal. Analytical methods exist which allow $\mathbf{J}_{uq}$ to be calculated from the spatial force derivatives (energy Hessian), to full accuracy and in much less time, with the total calculation being O(n^2). Note that this is within a constant factor of optimal for filling in a matrix with n^2 elements.

If possible, Simbody 1.0 should include a good modern method for calculating **J** analytically, but if that can't be done it should at least provide an interface designed to support such a calculation in the next release.

For implicit integration the required matrix is the full **J** (with nonzero velocities) rather than just $\mathbf{J}_{uq}$. However, that is not much worse. Calculating the $\mathbf{J}_{uq}$ submatrix is by far the most difficult part since it involves the Hessian of the potential forces and (formally) the partial derivative of the mass matrix *inverse* with respect to the q's.

8.6 Root finding and optimization

The needed computations here depend on the kind of problems being solved. They typically require Jacobians of various calculations with respect to the generalized coordinates and speeds. **J** as defined above can be very useful for minimizations involving search for equilibria. For satisfying constraints, the partial derivatives of the constraint equations 4de are required. Simbody 1.0 should provide access to these matrices, which are needed internally anyway.

Root finding problems can be difficult when the coordinates are constrained, so it is convenient to define a new set of fully-independent coordinates. In particular, Simbody 1.0 should do this at least for the case where the only constraints are the quaternion normalization conditions. It is easy to create a localized 3-coordinate representation for orientation about a current set of q's which will remain valid even for large perturbations. Reduced sets of coordinates for more general constraints may have limited validity ranges and have to be recalculated periodically during a root finding or optimization run.

Acknowledgments

I thank Dan Rosenthal for patiently teaching me everything I know (plus much more which I promptly forgot) about the fascinating field of multibody dynamics, and Linda Petzold for teaching me what little I know about the equally fascinating field of numerical integration and specifically how to solve Equation 4.

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.

References

[1] Schwieters, CD; Clore, GM. Internal Coordinates for Molecular Dynamics and Minimization in Structure Determination and Refinement. *J. Magnetic Resonance* 152:288-302 (2001).

[2] Chang, KS; Khatib, O. Efficient Algorithm for Extended Operational Space Inertia Matrix. *Proc. of the 1999 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems* (1999).

[3] Ascher, UM; Chin, H; Petzold, LR; Reich, S. Stabilization of constrained mechanical systems with DAEs and invariant manifolds. *Mechanics of Structures and Machines* 23(2):135-157 (1995).

[4] Eich, E. Convergence results for a coordinate projection method applied to mechanical systems with algebraic constraints. SIAM J. on Numerical Analysis 30(5):1467-1482 (1993).