Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2013)
J. Bender, J. Dequidt, C. Duriez, and G. Zachmann (Editors)

RPI-MATLAB-Simulator: A tool for efficient research and practical teaching in multibody dynamics

J. Williams[1] & Y. Lu[1] & S. Niebe[2] & M. Andersen[2] & K. Erleben[2] & J.C. Trinkle[1]

[1]Rensselaer Polytechnic Institute, 110 8^{th} Street, Troy, NY 12180
[2]University of Copenhagen, Universitetsparken 5, DK-2100 Copenhagen

Abstract

We present the RPI-MATLAB-Simulator (RPIsim) as an open source tool for research and education in multibody dynamics. RPIsim is designed and organized to be extended. Its modular design allows users to edit or add new components without worrying about extra implementation details. RPIsim has two main goals: 1. Provide an intuitive and easily extendable platform for research and education in multibody dynamics; 2. Maintain an evolving code base of useful algorithms and analysis tools for multibody dynamics problems. Although research often focuses on a specific subset of problems, work too often begins with developing software in a broader scope simply to realize a test bed for research to begin. It is our hope that RPIsim alleviates some of this burden by decreasing development time, thusly increasing efficiency in research. Further, we aim to provide a practical teaching tool. Because it is a fully working simulator, and since it offers the instant gratification of visualized contact dynamics, RPIsim offers students the opportunity to experiment and explore dynamics in the powerful environment of MATLAB. With multiple built-in simulation methods, and support for a simulation data convention, RPIsim facilitates the fair comparison of methods, including those being developed with RPIsim.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Physically based modeling—D.4.8 [Software]: Performance—Simulation D.2.8 [Software]: Metrics—Performance measures I.6.1 [Computing Methodologies]: Simulation and Modeling—Simulation Theory

1. INTRODUCTION

Since the late 1980s, many physics engines have been developed - some open source, some proprietary. Many impressive videos available online make it appear as though the problems are all solved, but this is far from the case. In fact, poor performance in terms of simulation speed and physical fidelity continue to prevent application of physical simulation in many domains, for example, model predictive control and state estimation in robotics. The desire for greater simulation performance drives research on different ways to formulate the simulation equations and solve them [BETC12, TET12], and on calibrating and tuning simulations to best match physical observations of the real systems we simulate [ZBT10, BNT09, PG96]. An important topic which is nearly unexplored is the domain of applicability or region of trust of a simulation. More specifically, determining over what set of physical model parameters, driving inputs, and solver parameters, will a simulation be "valid" or "correct enough" to enable a chosen application.

One of the primary goals of the RPI-MATLAB-Simulator (RPIsim) is to provide an intuitive, easily extendable platform to support research and education in multibody simulation. A design goal that has made RPIsim unique is the goal to compare, in an unbiased manner, many simulation methods. The motivation for this is the plethora of research papers that explain and demonstrate their simulation method, but do not (or cannot for lack of a convenient tool) compare it fairly to existing simulation methods. RPIsim currently supports a database of test problems stored in a format that allows the construction of many different time-stepping subproblems (a subproblem is the system of equations and inequalities that must be formulated and solved at every time step to advance the simulation in time). RPIsim also supports several different solution algorithms compatible with these formulations, and instrumentation at the solver iteration level for solver analysis.

In addition to the necessary further research in multibody dynamics, RPIsim provides a platform for education. The

level of interaction with the simulator can vary dramatically and begins with the very basic (section 5.1). Student tasks might include generating simple scenes and plotting position, velocity, and acceleration of sliding or falling bodies, or assignments could involve implementation of a linear complementarity problem solver. In both of these examples, RPIsim allows the student work to be focused since all other simulation functionality is in place.

RPIsim supports an accuracy enhancement supported by no other simulator in the world, which we refer to as polyhedral exact geometry (PEG). In all other simulators, a non-penetration constraint between pairs of bodies must be added to the time-stepping subproblem when the bodies are nearer to each other than some prescribed tolerance. When the nearest features on approaching bodies are vertices that are each locally convex from the object perspective, then the set of all non-penetrating relative velocities is non-convex. Despite this fact, existing simulators choose a locally convex approximation of the set, which causes non-physical simulation artifacts. We discuss this issue in more detail in section 3 and an illustrative example is given in section 5.1.

Improving the physical accuracy of simulation is an important objective. Consider the on-going DARPA Robotics Challenge (www.theroboticschallenge.org). In the challenge, a humanoid robot must get into a car, drive to a disaster site, use a jackhammer to break down a wall, climb a ladder, attach a hose to a standpipe, turn a valve, and more. The first part of the Challenge was the Virtual Challenge (held in June 2013), in which all of the tasks were done completely in simulation using the ROS/Gazebo simulator built on top of Open Dynamics Engine (ODE). The winners of the Virtual Challenge have received a physical Atlas robot that they will enter in a physical competition in Miami on December 21, 2013. The hope of the DARPA division that sponsored the competition is that the experience gained in the Virtual Challenge will transfer with minimal tweaking to the real-time controllers of the physical robot. In particular, teams will want to use the simulation in planning and controlling dynamic actions, like running across uneven terrain. In these cases, the simulation has to predict the robot's behavior accurately, because high-level plans will be chosen based on simulation outcomes.

It is too early to tell how smoothly the transition from virtual to real will go, but one author's team has already run into significant issues - all related to the trade-off between speed and accuracy in simulation. In our specific case, the problem is that the mass ratio between the smallest finger link to the torso link is on the order of 10^4, the simulation time step is fixed at 0.016 seconds, and the robot's controllers run at 1000 Hz. Further, an extremely important functionality of the robot is the ability to control the forces it applies to the objects it grasps, perhaps the roll cage of a simulated vehicle (this pushes the mass ratio much higher) or a tool. In both of these cases, closed kinematic loops with large internal forces are formed, making the simulation more challenging.

Consider Figure 1, which shows side-by-side frames of simulations produced in Blender using Bullet (left column) whose physics engine is a derivative of ODE's, and a slower but more accurate engine, dVC3d (right) [Ngu11]. Time advances down the page. The accompanying videos are available from http://www.youtube.com/watch?v=pj49NKW6n8U and http://www.youtube.com/watch?v=qx6GjnLnf5Q. Problems similar to those revealed in the Bullet video will arise in any simulator when pushed hard enough.

In these two grasp simulations, the Barrett Hand moves toward the chalice and closes the fingers. In the right column, all goes as expected, but in the left column problems are already visible in the second frame - the chalice is not touching the block. The next frame shows the fingers on the back side of the chalice splaying due to joint constraint errors, which the fourth frame shows most blatantly where the distal joint of the finger in full view is stretched apart and twisted. Eventually the chalice escapes the grip of the hand entirely due to instabilities. Imagine using simulation to plan to grasp the chalice. Suppose a robot was told to pick up the chalice and its knowledge of the physical world was the Bullet simulator as set up for this video comparison. In that case, the robot would decide that it was impossible to grasp the chalice, so it would fail.

Contributions.

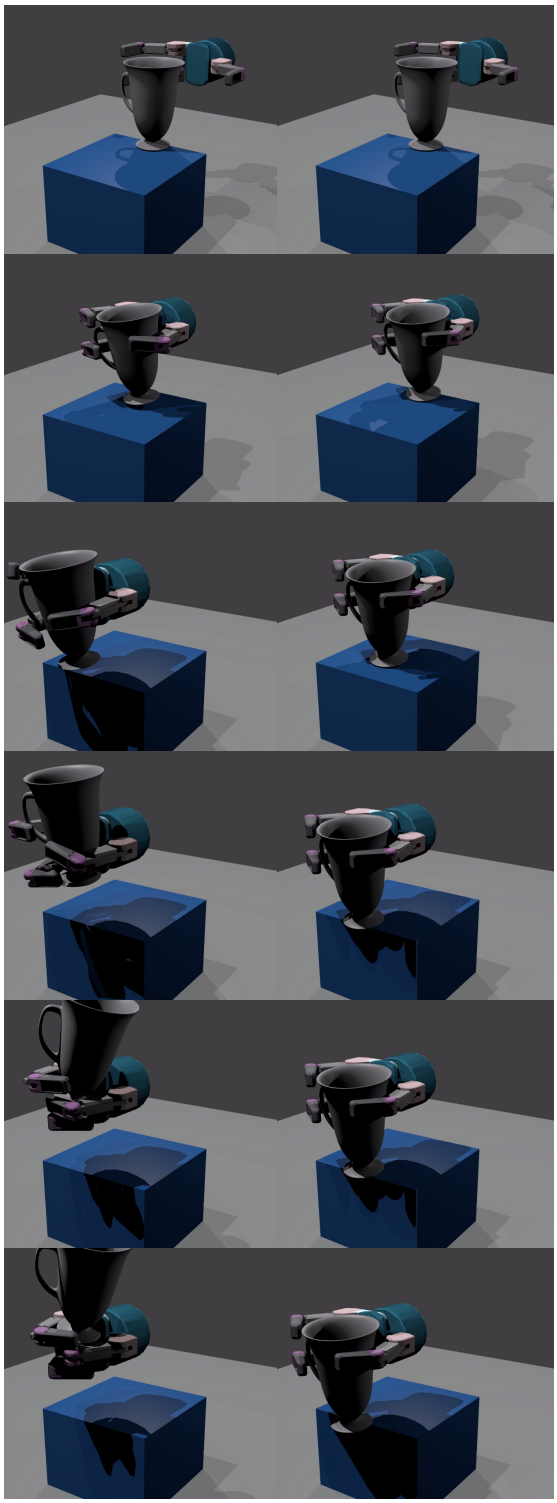
- An easily extendable open source simulator in MATLAB for research and education in multibody dynamics.
- Support for a simulation data convention that facilitates recording of simulation data, for example as benchmark problems for comparison of solver performance.
- A framework for simulation experiments, particularly useful for comparing time-stepping formulations.

2. SIMULATION OVERVIEW

MATLAB [MAT10] was chosen as the platform in which to build RPIsim because of its ease of use and wide availability (efforts have been made to achieve compatibility with Octave [Eat02]). MATLAB offers a huge body of functions and an environment that facilitates rapid development of mathematical-based software for research. This section introduces the structure of the simulator and the basic steps for adding custom modules.

2.1. Interface

A simulation is created by defining a set of bodies, adding these bodies to a *Simulation* structure, then starting the the simulation. Previous versions of RPIsim utilized a custom GUI for user-friendly interaction during simulation, however the most recent version of RPIsim has sacrificed this small



(a) Bullet physics (b) dVc3d

Figure 1: Comparison of stability in Bullet physics versus dVc3d time-stepping method in Blender.

convenience in order to achieve compatibility with Octave as well as improve timing performance during simulation. If the simulation is run with the GUI enabled (default), then the user is still able to navigate the scene using the mouse. When run without the GUI (more efficient), the user need only set the maximum number of time steps to define a stopping criterion.

The file structure of the simulator is depicted in Figure 2. This structure is meant to be intuitive and help guide the user when editing or adding new components. The "examples" directory contains several examples of how to create scenes, specify options, and run a simulation. The simulator itself is entirely contained in the "engine" directory. The directories found there are fairly self-explanatory. The "dynamics" directory contains the functions defining each available time-stepping formulation, all of which construct a time-stepping subproblem formulated as a complementarity problem (CP). The "solvers" directory contains various functions for solving the complementarity problem: linear complementarity problem (LCP), mixed linear complementarity problem (mLCP), and nonlinear complementarity problem (NCP). See [BETC12] for a comprehensive review of these topics.

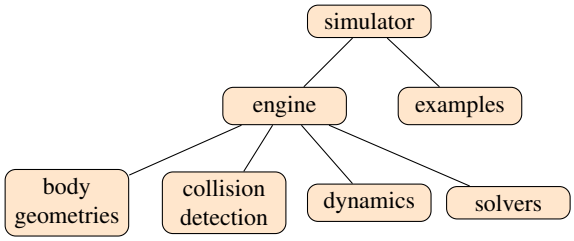


Figure 2: RPIsim file structure. Simulator code is organized in order to reflect the stages of simulation and to give the user intuition about the connectedness of these stages. The existing code serves as a set of templates for extending the simulator with custom modules.

2.2. Simulation Loop

The flow of simulation is depicted in Figure 3. As soon as a simulation script is executed, the scene is rendered so that it may be inspected. When run, the simulator proceeds through the various stages of the simulation loop.

The userFunction stage is an optional stage that allows the user to put in place a custom function such as a controller or functionality for plotting. A controller could be an explicit time-based position or velocity controller, or a proportional-integral-derivative (PID) controller for joint control of a robotic arm. Although all simulation variables are available and editable at this stage, it is recommended that bodies be controlled only by setting external forces and allowing the dynamics to solve for the next step. This is similar to the idea behind energy functions in simulation [WFB87].

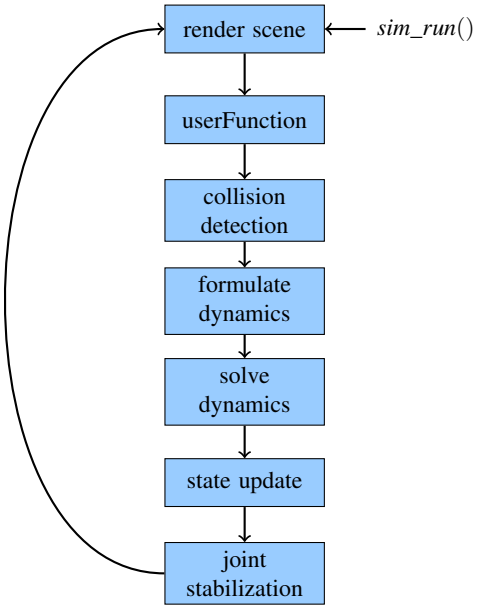


Figure 3: Simulation loop. At each stage, a user can replace or plug in custom modules.

The "formulate dynamics" stage constructs a time-stepping subproblem to be solved. The available formulations are described in section 3. This subproblem takes into account any forces generated in the previous stage as well as all non-penetration, friction, and joint constraints.

2.3. Modularity of Simulation Components

The modularity of RPIsim is simple because all simulation data is stored in a single *Simulation* structure, and function handles are used to call the various simulation stages. This means that any custom component need only worry about where to access relevant information for their component to process it, and which fields need to be updated before returning. For example, custom collision detection need only look at the list of bodies, understand the *Body* structure, and add contacts to the simulator before returning.

Details about adding custom modules are given in the following sections. In general, a simulation module is a function with a single parameter which is the *Simulation* structure. Within the simulator, adding a module simply involves setting the function handle to be executed at a given stage.

Adding Custom Collision Detection

Here we will reference mesh bodies as an example, but custom body geometries can be added by extending the *Body* structure. *Body* contains the basic kinematic attributes of a simulation body including position, velocity, and external forces.

If we wish to write a custom collision detection function, *newCD(sim)*, we start by creating a new function with that name which takes a *Simulation* structure as its only argument and returns the updated structure. We may then write *newCD* in terms of all the bodies within the simulation. In addition to the standard body information, mesh bodies contain all of the information about world vertex coordinates, as well as edges and faces in terms of indexed vertices.

Adding contacts to the current contact set of a Simulation struct *sim* is done by updating *sim.contacts*, an array of *Contact* structures. Each contact is a 5-tuple $(B_{id1}, B_{id2}, \boldsymbol{\pi}_1, \hat{\mathbf{n}}, \psi_n)$, where B_{id1} and B_{id2} are the body IDs of the bodies in contact, $\boldsymbol{\pi}_1$ is the point of contact in world space on the first body, $\hat{\mathbf{n}}$ is the unit normal direction of the contact from the first body, and ψ_n is the gap distance. By "gap distance," we refer to the signed distance between the two bodies at the contact. When in penetration, this value is negative, at the exact moment of contact this value is zero, and when near but not in penetration, this value is positive. It is important to be able to identify potential contacts within a given small distance ε if we wish to generate constraints that will prevent penetration.

Since extending RPIsim is designed to be straightforward, incorporating the new collision detection routine is as simple as setting the appropriate function handle in the simulator with *sim.H_collision_detection* = @*newCD* before running the simulator.

Adding Custom Dynamics Formulations

By default, a function *preDynamics()* is called before the "formulate dynamics" stage of Figure 3 which constructs common submatrices for all bodies found to be in contact, descriptions of which are given in section 3. These matrices are all stored in a struct called *dynamics* within the *Simulation* structure. A custom dynamics formulation may wish to use these values or choose to ignore them.

The format of the formulation is dependent on which solver will be used (and *vice versa*), but most solvers will be solving the LCP for a solution vector $\mathbf{z} \in \mathbb{R}^n$ where

$$\begin{aligned} \mathbf{z} &\geq \mathbf{0} \\ \mathbf{A}\mathbf{z}+\mathbf{b} &\geq \mathbf{0} \\ \mathbf{z}^T(\mathbf{A}\mathbf{z}+\mathbf{b}) &= \mathbf{0} \end{aligned} \tag{1}$$

where $\mathbf{z}^T$ denotes the transpose of $\mathbf{z}$. For such a problem, the dynamics formulation need only supply a suitable matrix $\mathbf{A}$ and vector $\mathbf{b}$. This is done by storing the values in the dynamics struct i.e. *dynamics.A* and *dynamics.b*.

Given a custom dynamics function *newDynamics*(*sim*), we incorporate the custom function by setting *sim.H_dynamics* = @*newDynamics*.

Adding Custom Solvers

A custom solver will likely be operating on the problem stored in the previously described dynamics struct. The im-

portant aspect of a custom solver is that it must return a set of new velocities $\mathbf{v}^{\ell+1}$ for each body that was active in the formulation. These new velocities are used in the next step of updating the state of all simulation bodies before restarting the simulation loop.

Incorporating a new solver simply involves setting *sim.H_solver = @newSolver*. All contact information at every simulation step is available in *sim.contacts*, and there are several solvers already in place that exemplify this access.

Alternative Dynamics

If one wishes to use an alternative dynamical method that does not fit the paradigm described above and depicted in Figure 3, it is possible to bypass the stage "formulate dynamics" and incorporate the alternative method entirely in a custom solver. This could be done for example with a penalty method where forces are determined solely on penetration depth without the need to call a solver.

2.4. Joints

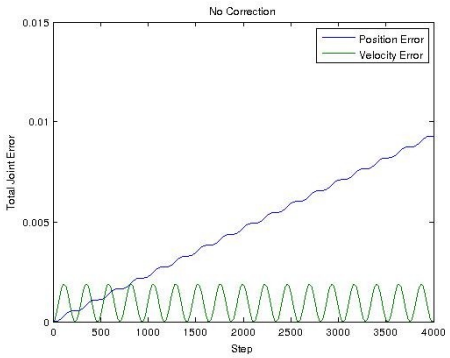
Joints are currently implemented in RPIsim as bilateral constraints between two bodies. A joint is defined by the two bodies it joins, the type of joint, and an initial position and joint direction.

When using the maximal coordinate formulation (see [Bar96]), joints are known to "drift" due to extra degrees of freedom. Joint stabilization based on the work in [BS06] is included in RPIsim in order to maintain joint constraints. Figure 4 shows the position and velocity errors for simulation of a hanging pendulum with time step of 0.01 seconds for simulations with and without joint correction. When stabilization was used, joint position and velocity errors were both bounded by an epsilon of 10^{-5}. The PATH solver [DF94] was used to solve the dynamics.

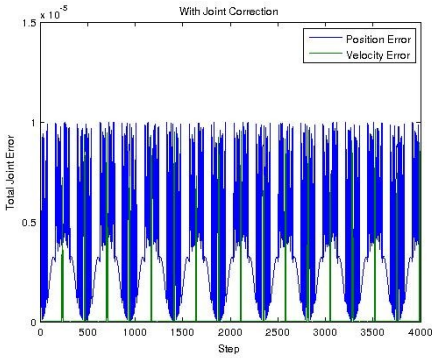
With minimal coordinates, bilateral constraints are eliminated using joint coordinates that directly parametrize the possible motion [APC95]. Thus minimal coordinate models have no bilateral constraints since they are implicitly incorporated in the Newton-Euler equation. Subsequently, no joint correction is needed for minimal coordinate formulations. We are currently in the middle of adding the minimal coordinate formulation to RPISim. Maximal coordinate formulations, already implemented, are consistent with both unilateral and bilateral constraints.

3. TIME STEPPING FORMULATIONS

RPIsim currently utilizes time stepping formulations, however we will briefly mention the event-driven integration method which uses a standard ODE integrator in the smooth phase of the system and a LCP, mLCP, NCP or augmented Lagrangian method to determine the next time when mode switching occurs. Various index sets, such as detaching,



(a) Joint errors without stabilization.



(b) Joint errors with stabilization ($\varepsilon = 10^{-5}$).

Figure 4: Joint errors for a hanging pendulum. Without stabilization, the pendulum "drifts." Note that the period of oscillation remains unchanged.

sticking, sliding are used to describe the kinematic state of contact points. The sets are not constant since the contact configuration of the dynamical system changes with time due to stick-slip transitions, impact, and contact loss. At an event, index sets are adjusted which then set up an LCP, and the new contact configuration is determined by this LCP [LN04].

To contrast, the event-driven method integrates the system until an event occurs, calculates the next mode, adjusts the set indices and proceeds integration. Time-stepping methods are based on a time-discretization of generalized position and velocities and for each time step, multiple events might take place simultaneously. This is especially useful when one is interested in a system with many contact points or with a large number of events that might occur in short amounts of time [LN04]. It is in part due to these benefits that RPISim currently supports time-stepping methods right now.

RPIsim has several built-in time-stepping formulations. The following subsections detail three of these formulations, all of which are derived from similar ideas in discrete dy-

namics. The literature on these topics is vast, however we will preface these descriptions with a brief description of their general background.

We begin with the Newton-Euler equation

$$\mathbf{M}\dot{\boldsymbol{\nu}} = \mathbf{G}\boldsymbol{\lambda} + \boldsymbol{\lambda}_{ext} \tag{2}$$

where $\mathbf{M}$ is the mass-inertia matrix diagonally composed of $\mathbf{M}_i = \begin{bmatrix} m_i\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_i \end{bmatrix}$ for each body i with mass m_i and inertia tensor $\mathbf{J}_i$, $\dot{\boldsymbol{\nu}}$ is the first derivative of the generalized velocities $\boldsymbol{\nu}_i = \begin{bmatrix} \mathbf{v}_i^T & \boldsymbol{\omega}_i^T \end{bmatrix}^T$, $\boldsymbol{\lambda}$ is the forces resulting from constraints imposed upon the bodies, $\mathbf{G}$ is the corresponding constraint Jacobian, and $\boldsymbol{\lambda}_{ext}$ is the external forces applied to the bodies. $\mathbf{M}$ and $\mathbf{G}$ are expressed in the world frame and therefore functions of the body configurations $\mathbf{q}$. That is, they are more accurately written as $\mathbf{M}(\mathbf{q})$ and $\mathbf{G}(\mathbf{q})$, but we shall abbreviate to simplify notation. We discretize equation (2) from time stepℓ to $\ell+1$ as

$$\mathbf{M}\boldsymbol{\nu}^{\ell+1} = \mathbf{M}\boldsymbol{\nu}^{\ell} + \mathbf{G}\mathbf{p}^{\ell+1} + \mathbf{p}_{ext}^{\ell+1} \tag{3}$$

where $\mathbf{p}$ and $\mathbf{p}_{ext}$ are impulses i.e. $\mathbf{p} = h\boldsymbol{\lambda}$ for time step size h.

We may separate the term $\mathbf{Gp}$ into distinct constraints $\mathbf{Gp} = \mathbf{G}_n\mathbf{p}_n + \mathbf{G}_f\mathbf{p}_f$, where $\mathbf{G}_n$ is the non-penetration constraint Jacobian, $\mathbf{p}_n$ is the vector of impulses applied at contact points in the normal directions of those contacts, $\mathbf{G}_f$ is the friction constraint Jacobian, and $\mathbf{p}_f$ is the vector of impulses applied perpendicular to contact normals at contact points due to friction (we are neglecting bilateral constraints, for now). Consider Figure 5 where a contact is defined by a

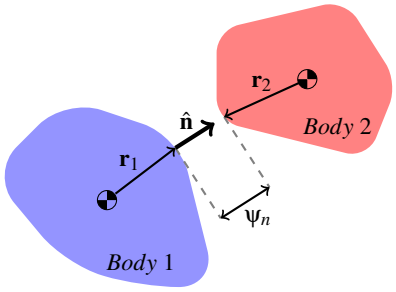


Figure 5: Two bodies and contact vectors.

point on each body, a normal direction, and a gap distance ψ_n. The non-penetration constraint Jacobian $\mathbf{G}_n$ is composed of submatrices $\mathbf{G}_{n_{ij}}$ over the i^{th} contact and j^{th} body where

$$\mathbf{G}_{n_{ij}} = \begin{bmatrix} \hat{\mathbf{n}}_i \\ \mathbf{r}_{ij} \times \hat{\mathbf{n}}_i \end{bmatrix}. \tag{4}$$

If the j^{th} body is the second body in the contact, then the normal $\hat{\mathbf{n}}_i$ is negated. The friction constraint Jacobian is composed of submatrices $\mathbf{G}_{f_{ij}}$ for n_d friction directions in the linearized friction cone where

$$\mathbf{G}_{f_{ij}} = \begin{bmatrix} \hat{\mathbf{d}}_{i_1} & \dots & \hat{\mathbf{d}}_{i_{nd}} \\ (\mathbf{r}_{ij} \times \hat{\mathbf{d}}_{i_1}) & \dots & (\mathbf{r}_{ij} \times \hat{\mathbf{d}}_{i_{nd}}) \end{bmatrix} \tag{5}$$

where $\hat{\mathbf{d}}_{ik}$ is the k^{th} vector representing the friction cone depicted in Figure 6.

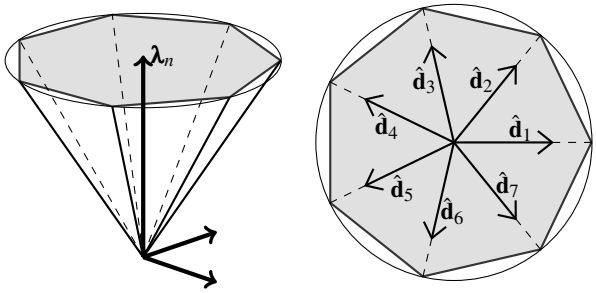


Figure 6: The friction cone and its polygonal approximation for $n_d = 7$ friction directions.

It should be noted that the time-stepping methods currently included with RPIsim are "preventative" methods in which inter-penetration between rigid bodies is ideally avoided, as opposed to "corrective" methods which wait for penetration to occur and then correct it. This type of preventative rigid body interaction is inelastic, i.e. does not include any "bounce" on contact [Ste00]. However, there are numerous ways to incorporate elastic collisions into a simulation with RPIsim. Perhaps the most straight-forward approach is to add a custom collision detection routine which only reports contacts from penetrations. Forces may then be generated based on energy functions or penalty methods.

3.1. Polygonally Exact Geometry

The most general dynamics formulation currently included in RPIsim is the polygonally exact geometry (PEG) time-stepping method. PEG is introduced in [NT10] and is derived and thoroughly covered in [Ngu11]. To give a brief explanation of the motivation, consider the case where the vertex v of one body is near an edge of another body, with potential contacts with faces f_1 and f_2 with distances of ψ_1 and ψ_2, respectfully. The 2D projection of this simple case is depicted in Figure 7. If the collision detection routine were to include both of these contacts, then time-stepping methods that attempt to enforce all non-penetration constraints independently, e.g. [ST96, AP97], would result in an erroneously applied impulse due to the negative gap distance of ψ_1. PEG groups certain sets of contacts into subcontacts such that one subcontact per set is enforced. This would allow the vertex v to accurately pass near the edge (or corner in 2D) in Figure 7. In this particular example, we would refer to a single contact with two subcontacts.

The PEG formulation currently implemented in RPIsim

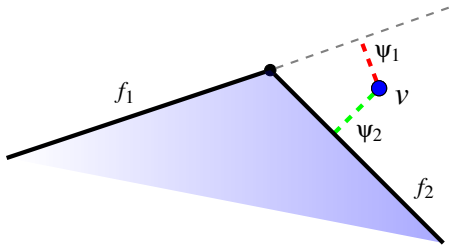


Figure 7: Vertex-face case. Consider the negative distance ψ_1 and how it is "corrected" if included.

uses a heuristic to choose a primary subcontact when there is more than one. If every contact has only a single subcontact, then the formulation reduces to the Stewart-Trinkle formulation [ST96]. Here, we will formulate PEG as a mixed LCP. It is considered "mixed" because of the inclusion of the Newton-Euler equation in a manner that requires equality with zero, and is does not take the form of a pure LCP as in equation (7). For n_b bodies, n_c contacts, n_s subcontacts, and n_d friction directions, the mLCP formulation of PEG is written as

$$\begin{vmatrix} \mathbf{0} \\ \boldsymbol{\rho}_n^{\ell+1} \\ \mathbf{a}^{\ell+1} \\ \boldsymbol{\rho}_f^{\ell+1} \\ \boldsymbol{\sigma}^{\ell+1} \end{vmatrix} = \begin{vmatrix} \mathbf{M} & -\mathbf{G}_n & \mathbf{0} & -\mathbf{G}_f & \mathbf{0} \\ \mathbf{G}_n^T & \mathbf{0} & \mathbf{E}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_a^T & \mathbf{0} & \mathbf{E}_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{G}_f^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E} \\ \mathbf{0} & \mathbf{U} & \mathbf{0} & -\mathbf{E}^T & \mathbf{0} \end{vmatrix} \begin{vmatrix} \mathbf{v}^{\ell+1} \\ \mathbf{p}_n^{\ell+1} \\ \mathbf{c}_a^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{vmatrix} + \begin{vmatrix} -\mathbf{M}\mathbf{v}^{\ell} - \mathbf{p}_{ext}^{\ell} \\ \frac{\boldsymbol{\psi}_n^{\ell}}{h} + \frac{\partial \boldsymbol{\psi}_n^{\ell}}{\partial t} \\ \frac{\boldsymbol{\psi}_a}{h} \\ \frac{\partial \boldsymbol{\psi}_f}{\partial t} \\ \mathbf{0} \end{vmatrix} \tag{6}$$

where $\mathbf{M}$ is the mass-inertia matrix, $\mathbf{G}_n$ and $\mathbf{G}_f$ are the penetration and frictional constraint Jacobians, respectfully, which contain only primary subcontacts, $\mathbf{U}$ is the diagonal matrix of coefficients of friction per contact, $\mathbf{v}$ is the vector of generalized velocities of bodies with contacts, $\boldsymbol{\psi}_n$ is the vector of gap distances for all primary subcontacts, $\mathbf{G}_a$ is an auxiliary matrix composed of submatrices over the i^{th} contact and j^{th} body where $\mathbf{G}_{a_{ij}}^T$ is defined as

$$\mathbf{G}_{a_{ij}}^T = \begin{bmatrix} \mathbf{G}_{n_1}^T - \mathbf{G}_{n_2}^T \\ \vdots \\ \mathbf{G}_{n_1}^T - \mathbf{G}_{n_{s_i}}^T \end{bmatrix},$$

$\boldsymbol{\psi}_a$ is the corresponding auxiliary vector composed of stacked subvectors

$$\boldsymbol{\psi}_{a_i} = \begin{bmatrix} \psi_1 - \psi_2 \\ \psi_1 - \psi_3 \\ \vdots \\ \psi_1 - \psi_{n_{s_i}} \end{bmatrix},$$

$$\mathbf{E} = blockdiag(\mathbf{e}_1, \ldots, \mathbf{e}_{n_c}) \text{ where } \mathbf{e}_i = ones(n_d, 1),$$

$$\mathbf{E}_1 = blockdiag(\mathbf{e}_{1_1}, \ldots, \mathbf{e}_{1_{n_c}}) \text{ where } \mathbf{e}_{1_i} = ones(n_{s_i} - 1, 1),$$

and

$$\mathbf{E}_2 = blockdiag(\mathbf{E}_{2_1}, \ldots, \mathbf{E}_{2_{n_c}}) \text{ where } \mathbf{E}_{2_i} = tril(ones(n_{s_i} - 1)).$$

The variables $\boldsymbol{\rho}_n$, $\mathbf{a}$, $\boldsymbol{\rho}_f$, and $\boldsymbol{\sigma}$ are slack variables. The third row of equation (6) (along with $\mathbf{E}_1$) effectively allows a non-penetration constraint to be enforced on a single half-space out of a set.

3.2. Stewart-Trinkle

The time-stepping formulation described in [ST96] is obtained by removing the third row and $\mathbf{E_1}$ from equation (6). As a result, Stewart-Trinkle suffers from the problem previously described regarding Figure 7. The Stewart-Trinkle formulation may be written in the form of an LCP as

$$\mathbf{0} \leq \begin{vmatrix} \mathbf{G}_n^T\mathbf{M}^{-1}\mathbf{G}_n & \mathbf{G}_n^T\mathbf{M}^{-1}\mathbf{G}_f & \mathbf{0} \\ \mathbf{G}_f^T\mathbf{M}^{-1}\mathbf{G}_n & \mathbf{G}_f^T\mathbf{M}^{-1}\mathbf{G}_f & \mathbf{E} \\ \mathbf{U} & -\mathbf{E}^T & \mathbf{0} \end{vmatrix} \begin{vmatrix} \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{vmatrix} + \begin{vmatrix} \mathbf{G}_n^T(\mathbf{v}^{\ell} + \mathbf{M}^{-1}\mathbf{p}_{ext}^{\ell}) + \frac{\boldsymbol{\psi}_n^{\ell}}{h} + \frac{\partial \boldsymbol{\psi}_n^{\ell}}{\partial t} \\ \mathbf{G}_f^T(\mathbf{v}^{\ell} + \mathbf{M}^{-1}\mathbf{p}_{ext}^{\ell}) + \frac{\partial \boldsymbol{\psi}_f}{\partial t} \\ \mathbf{0} \end{vmatrix} \perp \begin{vmatrix} \mathbf{p}_n^{\ell+1} \\ \mathbf{p}_f^{\ell+1} \\ \mathbf{s}^{\ell+1} \end{vmatrix} \geq \mathbf{0} \tag{7}$$

where after a solution is found for $\mathbf{p}_n^{\ell+1}$ and $\mathbf{p}_f^{\ell+1}$, new velocities are calculated by

$$\mathbf{v}^{\ell+1} = \mathbf{v}^{\ell} + \mathbf{M}^{-1}\mathbf{G}_n\mathbf{p}_n^{\ell+1} + \mathbf{M}^{-1}\mathbf{G}_f\mathbf{p}_f^{\ell+1} + \mathbf{M}^{-1}\mathbf{p}_{ext}^{\ell} \tag{8}$$

3.3. Anitescu-Potra

The time stepping method described in [AP97] is quite similar to the Stewart-Trinkle method, and the formulation is obtained from equation (6) in the same way as Stewart-Trinkle with the addition of setting the values in the right most vector to $\mathbf{0}$ for all rows except the first. In addition to suffering from the same false-contact problem as Stewart-Trinkle, Anitescu-Potra is sensitively dependent on the tuning of numerical tolerances regarding inclusion of contacts.

4. SOLVERS

Several solvers, most of them open source, are included with RPIsim and listed in Table 1. The PATH solver [DF94] is a good general purpose solver, it allows one to use the mLCP formulation which is convenient compared with the LCP formulation. However, PATH does not scale well and is not robust in all cases. The splitting methods are attractive for interactive simulations but are known to be inaccurate. If a level of inaccuracy is acceptable, then these types of solvers are robust but require stabilization terms to counter large drifting errors in the solutions. The interior-point and Newton methods offer performance advantages over PATH as they are tailored for mLCP and LCP forms in rigid body dynamics. Further, they avoid the full matrix assembly and inversion of the Newton matrix by using iterative sub-solvers.

Hence such methods can be competitive with regard to performance as well as comparable to the accuracy and robustness of PATH.

Table 1: Solvers available with RPIsim.

Name	Applicable Model	Category
Lemke	LCP	pivotal method
PATH	mLCP	mixed pivotal and iterative method
PGS	LCP, mLCP	iterative method with projection
Projected Jacobi	LCP, mLCP	iterative method with projection
Fixed-point	LCP, NCP	iterative method
Interior-point	LCP	iterative method
Fischer-Newton	LCP, NCP	non-smooth Newton method with line search
Minmap-Newton	LCP, NCP	non-smooth Newton method with line search

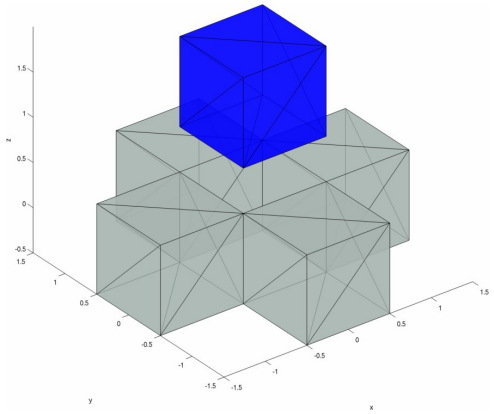
5. SIMULATION EXAMPLES

5.1. A "Hello World" Example

An example script for setting up a scene is shown in Figure 8, along with the scene that it generates. This simple script represents a benchmark 3D peg-in-hole problem. Four unit cubes are created, set as static bodies, and arranged such that the resultant space between them also has unit cube dimensions. A fifth cube is created, positioned, and scaled to 99.999%. This smaller dynamic cube should be able to pass through the gap "easily" with its 5μm clearance, however methods that anticipate collisions to prevent penetration (e.g. Stewart-Trinkle) will erroneously prevent the peg from entering unless special measures are taken to detect and remove contacts with the top faces of the static cubes. Conversely, methods that allow penetration may erroneously permit the peg to pass even if it were too large. Because RPIsim includes the 3D implementation of the PEG formulation, this simulation will give accurate results in all cases.

```
c1 = mesh_cube();
  c1.u = [-1;0;0];
  c1.dynamic = false;
c2 = mesh_cube();
  c2.u = [1;0;0];
  c2.dynamic = false;
c3 = mesh_cube();
  c3.u = [0;1;0];
  c3.dynamic = false;
c4 = mesh_cube();
  c4.u = [0;-1;0];
  c4.dynamic = false;
dropBox = mesh_cube();
  dropBox.u = [0;0;1.5];
  dropBox =
   scale_mesh(dropBox,0.99999);
bodies = [c1,c2,c3,c4,dropBox];
sim = Simulator();
sim = sim_addBody(sim, bodies);
sim = sim_run( sim );
```

(a) Script defining a sample scene.



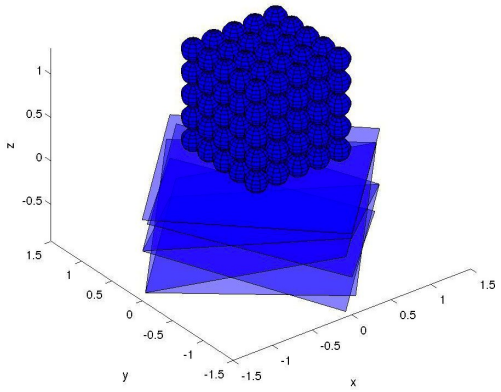
(b) 3D peg-in-hole simulation. Benchmark scenes such as this are important test cases for validating and assessing the physical accuracy of simulation methods. For example, virtual prototyping of robotics experiments requires proven accuracy if we hope to obtain useful results.

Figure 8: Sample script and the scene it generates.

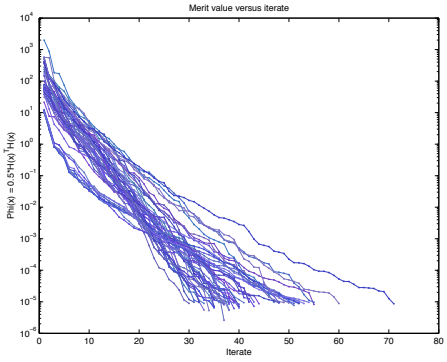
5.2. Analysis of Solvers

Developing, analyzing, and comparing numerical methods for solving the multibody complementarity problem, is made effortless by the inherent modularity of RPIsim. The scene shown in Figure 9a is borrowed from a research project, where it is used to benchmark solver performance in terms of accuracy, scalability and convergence rate. Using the natural merit function [AJMP11] on the solution vector from the solver, we can examine the convergence rate of the solver in question. For the test scene in Figure 9a, the natural merit function was computed at 50 samples over a total of 400 time-steps. Figure 9b shows the linear convergence rate achieved for the Fischer-Burmeister solver.

Other research is concerned not only with development and behavior of a single solver, but comparison of a set of solvers. In order to compare the performance of complementarity solvers, a data convention has been developed that utilizes the Hierarchical Data Format (HDF5) [LLWT13]. This



(a) Test scene of 125 spheres stacked in cube formation, used to test accuracy, scalability and convergence rate of the numerical solver.



(b) Fifty samples of convergence rate for Fischer-Burmeister solver. The natural merit function was used as an error metric for the solver.

Figure 9: Sample scene used to benchmark solvers.

convention is used to store simulation data including timing information, body information, and constraint violations at the level of single time steps. Such datasets are used to store benchmark problems and are useful for solver comparisons. Using RPIsim, a benchmark may be loaded and the time-stepping subproblem can be constructed using the variety of formulations available. These subproblems are then handed to the various compatible solvers and performance and errors may be easily compared. This is one of our active areas of research utilizing RPIsim, and involves assessment of fair metrics of accuracy and performance for sets of solvers that may differ dramatically.

5.3. Comparison of Time Stepping Formulations

Because RPIsim contains multiple time-stepping formulations, a framework is in place to compare the relative accuracy of these formulations against benchmark scenes. Such studies [FWT13b,FWT13a] have been done comparing Anitescu-Potra (AP) [AP97], Stewart-Trinkle (ST) [ST96], and PEG [Ngu11], and found that because both AP and ST have dependencies on parameter tuning of numerical tolerances involved in contact identification, PEG was more robust in many corner cases.

5.4. Grasp Experiments

Grasping is a significant area of research in robotics. Simulating grasping experiments requires accurate, robust, and stable dynamics. Unfortunately, it is difficult to find a simulator with these attributes. Grasping experiments are presently being researched with RPIsim. Figure 10 depicts a simulation experiment using the Schunk Powerball in RPIsim.

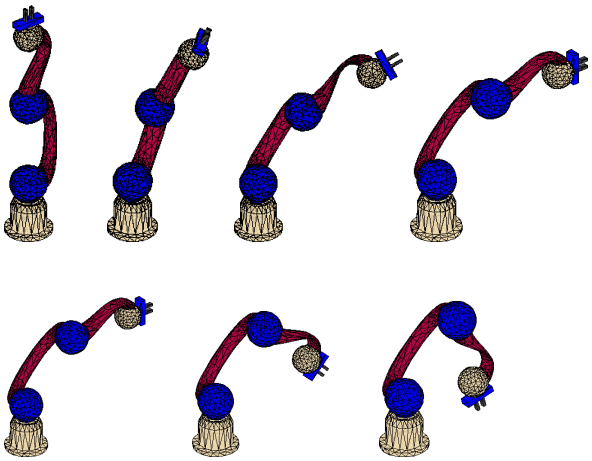


Figure 10: Simulation of Schunk Powerball arm executing a grasp trajectory. The trajectory is a set of joint angles interpolated between start and goal configurations over a given time.

6. SIMULATION UTILITIES

There are some utilities that improve usability of the simulator.

For users who have Bullet Physics [Cou] installed on their system, RPIsim offers an interface for passing polyhedral mesh data to Bullet collision detection through a MATLAB compiled MEX function. To utilize this feature, the user needs to compile the provided MEX function, and then set *useBULLET=true* in their *Simulation* structure.

By enabling recording of a simulation with $setRecord(true)$, a directory will be created upon execution where body properties and positions will be stored at every time step of simulation. This information can be animated and explored using the replay utility by calling $SimReplay(dir)$ where dir is the name of the new directory.

When a simulation includes joints, joint errors are recorded by default at every time step. This error is easily viewed after the simulation is completed by using $plotJointError(sim)$ where *sim* is the *Simulation* structure.

7. CONCLUSIONS & FUTURE WORK

We introduced the RPI-MATLAB-Simulator, a modular and extendable simulator written in MATLAB, for use in dynamics research and education. Some of the available features and utilities were described, including the available time-stepping formulations and solvers. Simulation examples were given, including active research using RPIsim.

There is still much to be done with RPIsim in terms of efficiency. As an environment using interpreted language, applications in MATLAB will never be as fast as their compiled counterparts. Optimization of data structures and algorithms in RPIsim is still underway. For example, the included collision detection routines are naive and will be updated to use hierarchical data structures and spatial and temporal coherence for retaining contact information.

RPI-MATLAB-Simulator is available at http://code.google.com/p/rpi-matlab-simulator/

7.1. Acknowledgements

Thanks to Claude Lacoursière for his insight and useful discussions. This work was partially supported by DARPA contract W15P7T-12-1-0002 and NSF grants CCF-0729161 and CCF-1208468. Any opinions, findings, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[AJMP11] ANDREANI R., JÚDICE J., MARTÍNEZ J., PATRÍCIO J.: On the natural merit function for solving complementarity problems. *Mathematical Programming 130*, 1 (2011), 211–223. 8

[AP97] ANITESCU M., POTRA F. A.: Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *NONLINEAR DYNAMICS 14* (1997), 231–247. 6, 7, 9

[APC95] ASCHER U. M., PAI D. K., CLOUTIER B. P.: Forward dynamics, elimination methods, and formulation stiffness in robot simulation, 1995. 5

[Bar96] BARAFF D.: Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 137–146. 5

[BETC12] BENDER J., ERLEBEN K., TRINKLE J., COUMINS E.: Interactive simulation of rigid body dynamics in computer graphics. In *Conference of the European Association for Computer Graphics, State of The Art Report (STAR)* (2012). 1, 3

[BNT09] BERARD S., NGUYEN B., TRINKLE J. C.: Sources of error in a rigid body simulation of rigid parts on a vibrating rigid plate. In *Proceedings of the 2009 ACM symposium on Applied Computing* (New York, NY, USA, 2009), SAC '09, ACM, pp. 1181–1185. 1

[BS06] BENDER J., SCHMITT A.: Fast dynamic simulation of multi-body systems using impulses. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Madrid (Spain), Nov. 2006), pp. 81–90. 5

[Cou] COUMANS E.: Bullet physics library. https://code.google.com/p/bullet/. An open source collision detection and physics library. 9

[DF94] DIRKSE S. P., FERRIS M. C.: The path solver for complementarity problems, 1994. 5, 7

[Eat02] EATON J. W.: *GNU Octave Manual*. Network Theory Limited, 2002. 2

[FWT13a] FLICKINGER D., WILLIAMS J., TRINKLE J.: Evaluating the performance of constraint formulations for multibody dynamics simulation. In *ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIEC)* (August 2013). Accepted. 9

[FWT13b] FLICKINGER D., WILLIAMS J., TRINKLE J.: What's wrong with collision detection in multibody dynamics simulation? In *IEEE International Conference on Robotics and Automation (ICRA)* (May 2013). Accepted. 9

[LLWT13] LACOURSIERE C., LU Y., WILLIAMS J., TRINKLE J.: Standard interface for data analysis of solvers in multibody dynamics. In *Canadian Conference on Nonlinear Solid Mechanics (CanCNSM)* (July 2013). 8

[LN04] LEINE R. I., NIJMEIJER H.: *Dynamics and Bifurcations of Non-Smooth Mechanical Systems*. Lecture Notes in Applied and Computational Mechanics. Springer, 2004. URL: http://books.google.com/books?id=CYxR-61vsMYC. 5

[MAT10] MATLAB: *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010. 2

[Ngu11] NGUYEN B.: *Locally Non-Convex Contact Models and Solution Methods for Accurate Physical Simulation in Robotics*. PhD thesis, Rensselaer Polytechnic Institute, Department of Computer Science, 2011. 2, 6, 9

[NT10] NGUYEN B., TRINKLE J.: Modeling non-convex configuration space using linear complementarity problems. In *IEEE International Conference on Robotics and Automation* (2010). 6

[PG96] PFEIFFER F., GLOCKER C.: *Multibody Dynamics with Unilateral Contacts*. Wiley, 1996. 1

[ST96] STEWART D., TRINKLE J.: An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal of Numerical Methods in Engineering 39* (1996), 2673–2691. 6, 7, 9

[Ste00] STEWART D. E.: Rigid-body dynamics with friction and impact. *SIAM Rev. 42*, 1 (Mar. 2000), 3–39. URL: http://dx.doi.org/10.1137/S0036144599360110, doi:10.1137/S0036144599360110. 6

[TET12] TODOROV E., EREZ T., TASSA Y.: Mujoco: A physics engine for model-based control. In *IROS* (2012), pp. 5026–5033. 1

[WFB87] WITKIN A., FLEISCHER K., BARR A.: Energy constraints on parameterized models. In *Computer Graphics* (1987), pp. 225–232. 3

[ZBT10] ZHANG L., BETZ J., TRINKLE J.: Comparison of simulated and experimental grasping actions in the plane. In *First Joint International Conference on Multibody System Dynamics* (May 2010). 1