2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Implicit nonlinear complementarity: A new approach to contact dynamics

Emanuel Todorov
Departments of Applied Mathematics and Computer Science & Engineering
University of Washington

Abstract— **Contact dynamics are commonly formulated as a linear complementarity problem. While this approach is superior to earlier spring-damper models, it can be inaccurate due to pyramid approximations to the friction cone, and inefficient due to lack of convexity coupled with a large number of auxiliary variables. Here we propose a new approach: implicit complementarity. Instead of treating contact velocities and forces as independent variables subject to explicit complementarity constraints, we express them as functions of a minimal set of unconstrained variables, and design these functions so that the complementarity constraints are automatically satisfied. We then solve the equations of motion via a non-smooth Gauss-Newton method augmented with an original linesearch procedure which exploits the problem structure. This enables us to represent the friction cone exactly and to reduce the number of unknowns by about a factor of 3. Numerical tests suggest that, in usage scenarios typical for robotics, the solver takes only about 5 iterations even without warm starts. More extensive tests and side-by-side comparisons remain to be done, but nevertheless the potential of the new approach is clear.**

I. Introduction

Modeling and simulation of contact phenomena is essential in a number of fields including robotics, mechanics and graphics. Earlier approaches were based on spring-damper models which often lacked stability and accuracy even after considerable manual tuning. More recently the physics of contact were cast as a linear complementarity problem (LCP). The latter approach has now become standard, not only in academia [1-6] but also in widely used simulation engines such as ODE, PhysX and Havok. Yet we are still far from having a solution which is both physically accurate and computationally efficient. This is because, even though the LCP formulation is sound, it is hard to handle algorithmically: direct solvers can be slow while iterative solvers can re-introduce many of the issues associated with earlier spring-damper models [2]. The goal of this paper is to develop a substantial modification to the LCP approach, improving both accuracy and efficiency.

Before delving into details we summarize the key idea. The contact impulse $\mathbf{f}$ and contact velocity $\mathbf{v}$ satisfy

$$A\mathbf{f} + \mathbf{v}_0 = \mathbf{v} \tag{1}$$

where the inverse inertia A and the bias velocity $\mathbf{v}_0$ are given. In the LCP approach, (1) is augmented with a set of complementarity constraints on $\mathbf{f}$ and $\mathbf{v}$. In our approach, we define functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ such that the complementarity constraints are satisfied for any $\mathbf{x}$, and then solve (1) with respect to $\mathbf{x}$ using non-smooth unconstrained optimization.

A. Review of the LCP approach

We begin by reviewing the basics of the LCP approach and introducing notation used later. Let $\dot{\mathbf{q}}, \ddot{\mathbf{q}} \in \mathbb{R}^{n_q}$ be the generalized velocity and acceleration of a rigid-body system with n_q degrees of freedom[1], $M(\mathbf{q})$ be the configuration-dependent inertia matrix, and $\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}})$ be all non-contact forces acting on the system including Coriolis, centripetal, gravitational and external forces. Suppose there are n_c contacts – found by a collision detection engine which is outside the scope of this paper. Let $\mathbf{n}_k \in \mathbb{R}^3$ be the unit vector normal to the two surfaces touching each other at contact k, and $\mathbf{t}_k^1, \mathbf{t}_k^2 \in \mathbb{R}^3$ be two orthogonal unit vectors spanning the tangent plane. Then $\Phi_k = \left[\mathbf{n}_k, \mathbf{t}_k^1, \mathbf{t}_k^2\right]$ is an orthonormal matrix defining the k-th contact coordinate frame. Let $\mathbf{v}_k, \boldsymbol{\lambda}_k \in \mathbb{R}^3$ be the contact velocities and interaction forces expressed in frame Φ_k, and $\mathbf{v}, \boldsymbol{\lambda} \in \mathbb{R}^{3n_c}$ be the stacked vectors of all contact velocities and forces. Let $K(\mathbf{q})$ be the Jacobian of the mapping from generalized coordinates to contact coordinates. Then the velocities $\mathbf{v}$ and $\dot{\mathbf{q}}$ are related as

$$K(\mathbf{q})\,\dot{\mathbf{q}} = \mathbf{v} \tag{2}$$

The contact forces expressed in generalized coordinates are $K(\mathbf{q})^{\mathsf{T}}\boldsymbol{\lambda}$, and so the equations of motion are

$$M(\mathbf{q})\,\ddot{\mathbf{q}} = \boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}) + K(\mathbf{q})^{\mathsf{T}}\boldsymbol{\lambda} \tag{3}$$

In the presence of Coulomb friction the above differential equations may not have a solution. This issue however is resolved if the dynamics are expressed in discrete time [1]. Let h be the time step and t be the time index. Replacing $\ddot{\mathbf{q}}_t$ with $(\dot{\mathbf{q}}_{t+h} - \dot{\mathbf{q}}_t)/h$, equations (2, 3) become

$$\begin{aligned} K(\mathbf{q}_t)\,\dot{\mathbf{q}}_{t+h} &= \mathbf{v}_{t+h} \\ M(\mathbf{q}_t)\,\dot{\mathbf{q}}_{t+h} &= M(\mathbf{q}_t)\,\dot{\mathbf{q}}_t + h\boldsymbol{\tau}(\mathbf{q}_t, \dot{\mathbf{q}}_t) + \\ &\quad hK(\mathbf{q}_t)^{\mathsf{T}}\boldsymbol{\lambda}_{t+h} \end{aligned} \tag{4}$$

This can be written more compactly as

$$\begin{aligned} K\dot{\mathbf{q}} &= \mathbf{v} \\ M\dot{\mathbf{q}} &= \mathbf{c} + K^{\mathsf{T}}\mathbf{f} \end{aligned} \tag{5}$$

where $\mathbf{c}_t = M_t\dot{\mathbf{q}}_t + h\boldsymbol{\tau}_t$ collects all the known terms in the second equation, and $\mathbf{f}_{t+h} = h\boldsymbol{\lambda}_{t+h}$ is the contact impulse.

Now the problem comes down to computing $\dot{\mathbf{q}}, \mathbf{v}, \mathbf{f}$ given $M, K, \mathbf{c}$. Even though (5) contains more unknowns than

[1]The development is very similar if we use redundant (Cartesian) coordinates and represent joints with equality constraints.

equations, the indeterminacy is resolved because $\mathbf{v}$ and $\mathbf{f}$ are further coupled through the laws of contact and friction. The latter can be formalized as follows. Suppressing the index k for clarity, let $\mathbf{f}$ be partitioned as $\left[f^{\mathcal{N}}; \mathbf{f}^{\mathcal{F}}\right]$ where $f^{\mathcal{N}}$ is the normal component and $\mathbf{f}^{\mathcal{F}} \in \mathbb{R}^2$ is the tangential/friction component, and similarly for $\mathbf{v}$. Along the normal we have

$$f^{\mathcal{N}} \geq 0, \quad v^{\mathcal{N}} \geq 0, \quad f^{\mathcal{N}} v^{\mathcal{N}} = 0 \tag{6}$$

These three conditions correspond to the fact that the contact impulse cannot pull the bodies towards each other, the bodies cannot penetrate, and if the contact is breaking then there can be no contact impulse. In the tangent plane we have

$$\begin{gathered} \mathbf{v}^{\mathcal{F}} \text{ parallel to } \mathbf{f}^{\mathcal{F}}, \quad \left\langle \mathbf{v}^{\mathcal{F}}, \mathbf{f}^{\mathcal{F}} \right\rangle \leq 0 \\ \left\| \mathbf{f}^{\mathcal{F}} \right\| \leq \mu f^{\mathcal{N}} \end{gathered} \tag{7}$$

The first line means that if there is slip then the friction force must act in the direction opposite to the slip velocity[2]. The second line means that the interaction force must lie inside the friction cone; μ is the coefficient of friction.

Condition (6) is called a complementarity condition. Together with a linear equation such as (5) it can form an LCP. Condition (7) on the other hand does not fit in the LCP framework because it involves nonlinearities. The standard approach is to approximate (7) by replacing the friction cone with a pyramid and allowing the slip velocity and friction force to be misaligned. This is done by choosing a redundant basis $\{\mathbf{d}_i\}$ for the tangent plane, where $\mathbf{d}_i$ are unit vectors that form a regular polygon. Assembling these vectors into the columns of the matrix D we can represent the friction impulse as $\mathbf{f}^{\mathcal{F}} = D\boldsymbol{\beta}$. After a few additional transformations which can be found in [1], the problem is approximated with an LCP. If the approximating friction pyramid is n_p-sided then the LCP involves $n_p + 2$ pairs of complementary variables per contact. In practice $n_p = 8$ is often used, resulting in $10n_c$ complementarity pairs.

B. Reasons to look for a new approach

We now discuss the algorithmic difficulties in solving the above LCP and explain the motivation behind our new approach. LCPs can be solved with direct pivoting methods such as Lemke's algorithm. Even though certain improvements to Lemke's algorithm have been developed specifically for the purpose of simulating contact dynamics [5][4], the algorithm remains slow. Indeed a recent analysis suggests that solving this LCP may be an NP-hard problem [2]. More precisely, it was shown in [2] that solving for the friction impulse given the normal impulse, or vice versa, corresponds to a convex quadratic program (QP). However solving for both simultaneously corresponds to a non-convex problem. The method proposed in [2] is to iterate between the two convex QPs. This is not guaranteed to converge, but in practice works quite well, especially with warm starts. A somewhat related simplification was proposed in [3] where the LCP was approximated with a convex problem – resulting in further inaccuracies in dealing with friction. Thus we already have algorithms which appear to be faster than Lemke's algorithm and more accurate than iterative solvers such as the Gauss-Seidel method included in most simulation engines. Yet the algorithmic difficulties and the proliferation of approximation schemes for a problem which is itself an approximation suggest that we should be able to find a better approach.

The more specific motivation behind our work is based on the following observations. The best general-purpose algorithm for solving LCPs is not Lemke's algorithm but the PATH algorithm [7]. Although the latter has rarely been used to simulate contact dynamics, our own experience has shown that it is indeed faster in this context. The original PATH algorithm was based on the path-search non-smooth Newton method [9]. The authors later found [8] that a simpler scheme works as well or better. This scheme is based on the Fischer-Burmeister function

$$\phi(a, b) = \sqrt{a^2 + b^2} - a - b \tag{8}$$

which has the property that $\phi(a, b) = 0$ if and only if a and b satisfy the complementarity condition (6). Thus, by replacing the complementarity condition with $\phi = 0$, one can convert a (linear or nonlinear) complementarity problem into a nonlinear system of equations. The resulting system is semi-smooth and can be solved with simpler generalizations of Newton's method [8][10].

In summary, the best algorithms for solving general LCPs are based on converting the LCP into a nonlinear system and applying some form of Newton's method. This raises two related questions. First, if the equation is going to be nonlinear anyway, why did we bother linearizing the friction cone? Second, could it be that we can arrive at a nonlinear equation more directly and intuitively, without going through an LCP and without introducing all the auxiliary variables $\boldsymbol{\beta}$ needed to approximate the friction cone? The latter point is particularly important because the computational bottleneck of all relevant algorithms is an inner loop solving linear equations repeatedly. A direct linear solver for a system with n equations takes $O\left(n^3\right)$ time. Thus, if we were to reduce the number of unknowns per contact from 10 to 3, this change alone could speed things up by a factor of 30. Below we develop a method which does just that.

II. Implicit nonlinear complementarity

The present results could be developed in either generalized or contact coordinates. The latter development is simpler, thus we focus on it throughout the paper. Since M is always invertible we can eliminate $\dot{\mathbf{q}}$ from (5) and obtain (1), where $A = KM^{-1}K^T$ is the inverse inertia matrix in contact coordinates while $\mathbf{v}_0 = KM^{-1}\mathbf{c}$ is the contact velocity which would result if $\mathbf{f} = 0$. Given A and $\mathbf{v}_0$, we seek $\mathbf{f}$ and $\mathbf{v}$ satisfying (1) along with (6, 7).

As stated in Introduction, our approach is to express both $\mathbf{f}$ and $\mathbf{v}$ as functions of some vector $\mathbf{x}$ with the same dimensionality, design these functions so that (6, 7) hold for any $\mathbf{x}$, and then solve (1) for $\mathbf{x}$. In other words we seek

[2]We consider the zero vector to be parallel to any other vector.

to convert problem (1, 6, 7) into a nonlinear equation of the form

$$A\mathbf{f}(\mathbf{x}) + \mathbf{v}_0 = \mathbf{v}(\mathbf{x}) \tag{9}$$

The remainder of this section will be devoted to constructing the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ which accomplish the above conversion. Algorithms for solving (9) will then be presented in the next section, followed by simulation results.

First we consider the normal direction. Focusing again on the case of a single contact, the functions in question are

$$\begin{aligned} f^{\mathcal{N}}(\mathbf{x}) &= \max\left(0,\ b - x^{\mathcal{N}}\right) \\ v^{\mathcal{N}}(\mathbf{x}) &= \max\left(b,\ x^{\mathcal{N}}\right) \end{aligned} \tag{10}$$

where b is a known constant. See **Fig 1A**. It is easiest to understand this construction in the case $b = 0$. When the scalar $x^{\mathcal{N}}$ is positive it encodes the normal velocity, otherwise it encodes the (opposite of the) normal impulse. The reason we can encode both $v^{\mathcal{N}}$ and $f^{\mathcal{N}}$ with one scalar is because they are complementary, thus only one of them can be non-zero. The variable $x^{\mathcal{N}}$ is "hybrid" in the sense that it has different units depending on its value, but this does not affect the math. The reason we need to allow $b \neq 0$ is because the complementarity condition (6) is actually somewhat restrictive. The more general condition is

$$f^{\mathcal{N}} \geq 0, \quad v^{\mathcal{N}} \geq b, \quad f^{\mathcal{N}}\left(v^{\mathcal{N}} - b\right) = 0 \tag{11}$$

This generalization is useful when two bodies have already penetrated and we want to push them apart – which can be achieved by using a positive b. Alternatively, if two bodies are not yet in contact but are moving towards each other, we may want to avoid penetration on the next time step – which can be achieved by using a negative b.

Next consider the tangent plane. Here things are more complicated because both $\mathbf{v}^{\mathcal{F}}$ and $\mathbf{f}^{\mathcal{F}}$ can be non-zero at the same time. However a compact representation is still possible because these two vectors are always parallel, thus their common direction can be encoded with the direction of the vector $\mathbf{x}^{\mathcal{F}}$. Then all we have to do is get the magnitudes right – which is similar to the complementarity encountered above. Define the scalar

$$s(\mathbf{x}) = \min\left(1,\ \frac{\mu f^{\mathcal{N}}(\mathbf{x})}{\|\mathbf{x}^{\mathcal{F}}\|}\right) \tag{12}$$

The functions we need can now be defined as

$$\begin{aligned} \mathbf{f}^{\mathcal{F}}(\mathbf{x}) &= -s(\mathbf{x})\,\mathbf{x}^{\mathcal{F}} \\ \mathbf{v}^{\mathcal{F}}(\mathbf{x}) &= \mathbf{x}^{\mathcal{F}} - s(\mathbf{x})\,\mathbf{x}^{\mathcal{F}} \end{aligned} \tag{13}$$

See **Fig 1B**. To understand this construction, first consider the case $s = 1$ which occurs when $\left\|\mathbf{x}^{\mathcal{F}}\right\| \leq \mu f^{\mathcal{N}}$. In this case equation (13) yields $\mathbf{f}^{\mathcal{F}} = -\mathbf{x}^{\mathcal{F}}$ and $\mathbf{v}^{\mathcal{F}} = 0$; in other words the interaction force is inside the friction cone and the contact is sticking. When $\left\|\mathbf{x}^{\mathcal{F}}\right\| > \mu f^{\mathcal{N}}$ we have $s < 1$. In this case the scalar s has the effect of scaling $\mathbf{x}^{\mathcal{F}}$ so that $\mathbf{f}^{\mathcal{F}} = -s\mathbf{x}^{\mathcal{F}}$ lies exactly on the friction cone. The "remainder" of the vector $\mathbf{x}^{\mathcal{F}}$ is then interpreted as the slip velocity. Note that the functions defined in (10) and (13) have the property

$$\mathbf{v}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \mathbf{x} \tag{14}$$

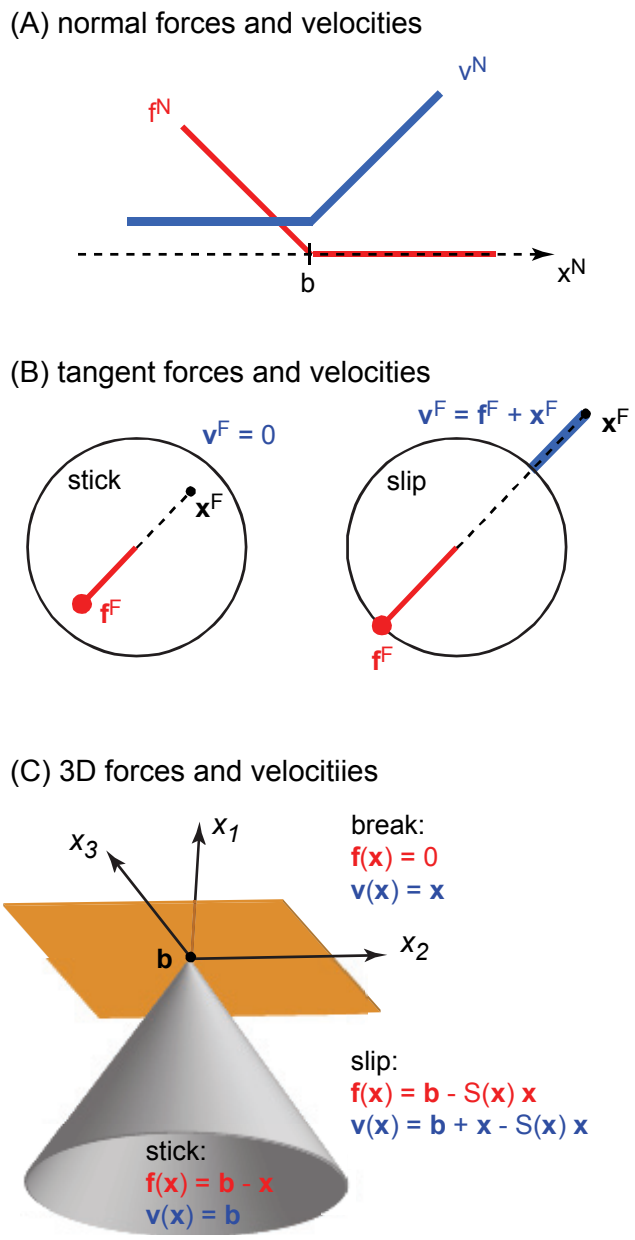

Fig. 1. Schematic illustration of the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$.

This is useful because we only need to compute $\mathbf{f}(\mathbf{x})$.

Recalling that $\mathbf{f} = \left[f^{\mathcal{N}}; \mathbf{f}^{\mathcal{F}}\right]$ and similarly for $\mathbf{v}$ and $\mathbf{x}$, we can now combine the normal and tangent spaces in the 3D representation illustrated in **Fig 1C**. The equations shown in the figure involve the vector $\mathbf{b} = [b; 0; 0]$ and the diagonal matrix $S(\mathbf{x}) = \mathrm{diag}(1, s(\mathbf{x}), s(\mathbf{x}))$. These equations are identical to (10) and (13), except we now see explicitly how they depend on the contact state (break, stick, slip) which is determined by the value of $\mathbf{x}$. The functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ are linear in the break and stick regions and nonlinear in the slip region. The nonlinearity is further illustrated in **Fig 2**, which shows how the first component of the tangent/friction impulse depends on $\mathbf{x}$ in two different planar sections of the 3D contact space.

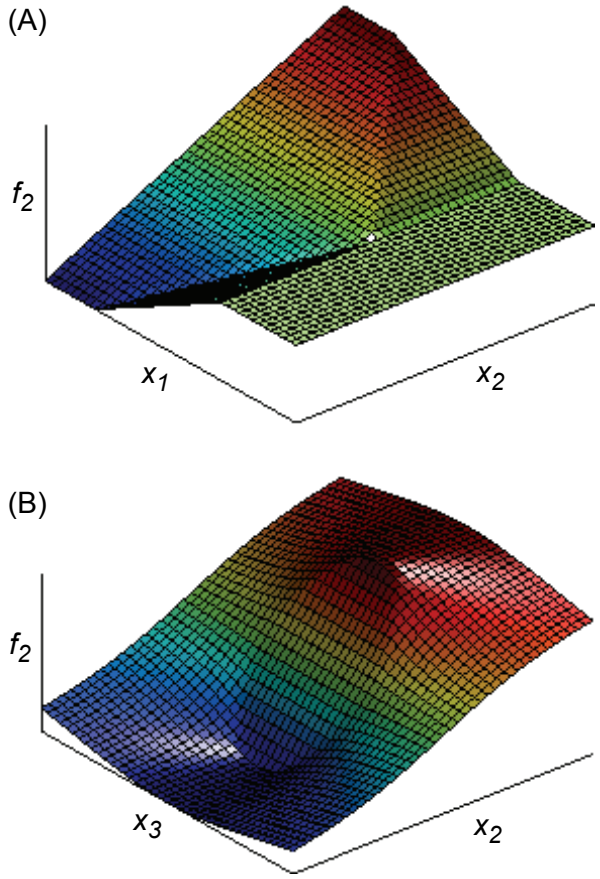

Fig. 2. Illustration of the nonlinearity in the function $\mathbf{f}(\mathbf{x})$ in two different planar sections of the 3D contact space.

III. Non-smooth Gauss-Newton method adapted to our problem

Having defined the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ we now proceed with algorithms for solving (9). Moving all terms to the left hand side and using (14), the residual is

$$\mathbf{r}(\mathbf{x}) = (A - I)\,\mathbf{f}(\mathbf{x}) - \mathbf{x} + \mathbf{v}_0 \tag{15}$$

The problem then reduces to solving the nonlinear equation

$$\mathbf{r}(\mathbf{x}) = 0 \tag{16}$$

where $\mathbf{x} \in \mathbb{R}^{3n_c}$ is unconstrained. Such problems are usually solved using Newton's method:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - J\left(\mathbf{x}^{(i)}\right)^{-1} \mathbf{r}\left(\mathbf{x}^{(i)}\right) \tag{17}$$

where i is the iteration number and J is the Jacobian:

$$J(\mathbf{x}) = \frac{\partial \mathbf{r}(\mathbf{x})}{\partial \mathbf{x}} \tag{18}$$

The same general idea will be applied here. It will turn out however that a number of important issues need to be addressed before the algorithm works efficiently. These issues include obvious ones such as instability and lack of smoothness, as well as less obvious ones having to do with chattering and loss of second-order convergence.

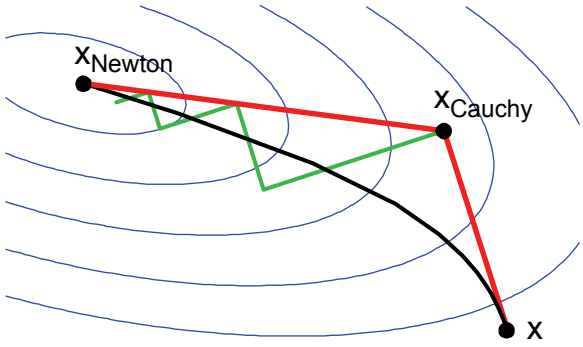

Fig. 3. Illustration of first and second-order optimization.

A. Review of second-order optimization

Here we explain the standard components of our algorithm while at the same time providing a brief review of relevant topics. It is well known that solving a nonlinear equation using Newton's method is equivalent to defining a sum-of-squares objective function

$$\ell(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^{\mathsf{T}}\,\mathbf{r}(\mathbf{x}) \tag{19}$$

and minimizing it with the Gauss-Newton method. The advantage of casting the problem in the optimization framework is that the function ℓ gives us a well-defined notion of progress, which is essential in designing procedures that enforce robustness. The gradient of ℓ is

$$\mathbf{g}(\mathbf{x}) = J(\mathbf{x})^{\mathsf{T}}\,\mathbf{r}(\mathbf{x}) \tag{20}$$

Define the matrix function

$$H(\mathbf{x}, \lambda) = J(\mathbf{x})^{\mathsf{T}} J(\mathbf{x}) + \lambda I \tag{21}$$

as well as the shortcut $H(\mathbf{x}) = H(\mathbf{x}, 0)$. This matrix is the Gauss-Newton approximation to the Hessian of ℓ; the exact Hessian includes an additional term dependent on the derivative of J. When J is invertible we have $J^{-1}\mathbf{r} = H^{-1}\mathbf{g}$, and so Newton's root-finding method is equivalent to the Gauss-Newton optimization method. However, when J is singular or close to singular, using $\lambda > 0$ makes the iteration more stable. This is the essence of the Levenberg-Marquardt algorithm which adapts λ online. Our implementation also uses such an adaptive λ.

Given the current setting of λ, define the "Newton point"

$$\mathbf{x}_{\mathrm{N}} = \mathbf{x} - H^{-1}\mathbf{g} \tag{22}$$

which is the minimum of the local quadratic (in $\boldsymbol{\epsilon}$) model

$$q(\mathbf{x} + \boldsymbol{\epsilon}) = \ell(\mathbf{x}) + \boldsymbol{\epsilon}^{\mathsf{T}}\mathbf{g}(\mathbf{x}) + \frac{1}{2}\boldsymbol{\epsilon}^{\mathsf{T}} H(\mathbf{x}, \lambda)\,\boldsymbol{\epsilon} \tag{23}$$

When $\ell(\mathbf{x}_{\mathrm{N}})$ is sufficiently smaller than $\ell(\mathbf{x})$ we accept $\mathbf{x}_{\mathrm{N}}$ as the next iterate. In our implementation $\mathbf{x}_{\mathrm{N}}$ is accepted if the actual improvement is at least 50% of the improvement predicted by the quadratic model. If this test fails, which happens when q is a poor approximation to ℓ, we need to backtrack in some way. The simplest approach is a

linesearch from $\mathbf{x}_{\mathrm{N}}$ to $\mathbf{x}$. This however may be problematic because, when q is a very poor model of ℓ and only small improvements are possible, the optimal search direction is given by the gradient. Backtracking along the curve $\mathbf{p}(\lambda) = \mathbf{x} - H(\mathbf{x},\lambda)^{-1}\mathbf{g}(\mathbf{x})$ would be ideal because this curve departs from $\mathbf{x}$ in the direction of the gradient and then turns towards the Newton point[3]. However such "curvesearch" would be computationally expensive. A common alternative, also used in our implementation, is the dogleg method. It relies on the "Cauchy point" defined as the minimum of (23) in the direction of the gradient $\mathbf{g}$:

$$\mathbf{x}_{\mathrm{C}} = \mathbf{x} - \mathbf{g}\frac{\mathbf{g}^{\mathsf{T}}\mathbf{g}}{\mathbf{g}^{\mathsf{T}}H\mathbf{g}} \tag{24}$$

The dogleg method consists of two linesearches: from $\mathbf{x}_{\mathrm{N}}$ to $\mathbf{x}_{\mathrm{C}}$, and from $\mathbf{x}_{\mathrm{C}}$ to $\mathbf{x}$. The typical configuration of these points is shown in **Fig 3**. The blue ellipses are the contours of the quadratic q. The curve $\mathbf{p}(\lambda)$ is shown in black. The green lines show the progress of a first-order steepest descent method (see below).

B. Extension to semi-smooth functions

In our case the function $\mathbf{r}(\mathbf{x})$ is continuous but non-smooth. One can verify that $\mathbf{r}(\mathbf{x})$ is semi-smooth, which is equivalent to having uniform convergence of all directional derivatives [11]. Dealing with semi-smooth functions is in principle straightforward. If the function is differentiable at the current $\mathbf{x}$ we proceed as in smooth optimization. If not, we form the set $\{J_m(\mathbf{x})\}$ of all possible Jacobians obtained by approaching $\mathbf{x}$ from different directions. The convex hull of this set is called the sub-differential. We can then pick any J inside this convex hull and use it in the above algorithm. The usual convergence properties are preserved [11][10][8].

Let us now be more specific about how we pick J when $\mathbf{r}(\mathbf{x})$ is non-differentiable. Recall that $\mathbf{x} \in \mathbb{R}^{3n_c}$ and that $\mathbf{x}_k \in \mathbb{R}^3$ is the part of $\mathbf{x}$ corresponding to the k-th contact. If any $\mathbf{x}_k$ lies on the plane or on the cone shown in **Fig 1C**, the residual $\mathbf{r}(\mathbf{x})$ is non-differentiable. We define an *edge* to be a $3n_c - 1$ dimensional manifold (plane or cone) where one of the contacts is in such a critical state. There are a total of $2n_c$ edges. Their intersections correspond to points where multiple contacts are in a critical state. We will ignore these intersections for simplicity of the exposition, although they are handled in a similar way. Suppose $\mathbf{x}$ lies on an edge, the Jacobians on the two sides of the edge are J_1 and J_2, and the normal vector to the edge is $\mathbf{d} \in \mathbb{R}^{3n_c}$ pointing towards the smooth region which corresponds to J_1. Then the directional derivatives of ℓ orthogonal to the edge are

$$\begin{aligned} \delta_1 &= \mathbf{r}^{\mathsf{T}} J_1 \mathbf{d} \\ \delta_2 &= -\mathbf{r}^{\mathsf{T}} J_2 \mathbf{d} \end{aligned} \tag{25}$$

If $\delta_1 < 0$ or $\delta_2 < 0$ then the objective function ℓ can be reduced in a direction which does not lie within the (tangent space to the) edge. In this case we pick the Jacobian corresponding to the smooth region which allows the steeper descent. Otherwise ℓ can only be reduced along the edge. We will call such an edge *active*. In this case we pick

$$J = \frac{\delta_2 J_1 + \delta_1 J_2}{\delta_1 + \delta_2} \tag{26}$$

It can be verified that this J is the unique element of the sub-differential for which $\mathbf{g} = J^{\mathsf{T}}\mathbf{r}$ lies within the edge.

Given the convergence guarantees of the generalized Gauss-Newton method for semi-smooth functions, and the fact that edges are a set of measure zero so we are unlikely to ever land on them anyway, we had expected the algorithm described above to always work well. Instead we found that it sometimes works well, but sometimes has slow convergence. Analysis of the problem revealed a chattering phenomenon, which led us to the improvements described in the next section.

C. Avoiding chattering through edge-aware linesearch

In the context of smooth optimization chattering is associated with plain (i.e. first-order) gradient descent methods. This is illustrated with the green curve in **Fig 3**. An exact line-search in the direction of the gradient yields a point where the gradient is orthogonal to the current search direction, causing a 90deg turn in each iteration. This chattering is the primary reason for using second-order methods such as Newton's method and its many variants.

However we are already using Newton's method here. So why do we get chattering? The reason is illustrated in **Fig 4A**, which shows the contours of two smooth functions joining at an edge. Suppose that each function is a perfect quadratic which can be minimized with a single Newton iteration. The problem is that the minimum of each quadratic lies in the domain of the other quadratic, and so a direct application of Newton's method will cause an infinite oscillation between the two virtual minima. When Newton's method is augmented with a linesearch it will backtrack somewhat, causing a decreasing sequence of oscillations which will eventually converge, but a lot of time will be wasted.

The above analysis makes it clear what the remedy is: if the line segment along which we are searching crosses an edge, check to see if the crossing is a local minimum, and if so accept it as the next $\mathbf{x}$. The algorithm will then automatically search along the edge on the next iteration, thanks to (26). We call this edge-aware linesearch. Of course this procedure is computationally efficient only if we can compute the line-edge intersections analytically. Fortunately this is the case here. For planar edges the computation is obvious. For cone edges we parametrize the line segment as $\mathbf{t}(\alpha) = \mathbf{x} + \alpha\mathbf{u}$ and then look for α such that $\mathbf{t}(\alpha)$ lies on the friction cone:

$$\mu^2 (t_1(\alpha) - b)^2 = t_2(\alpha)^2 + t_3(\alpha)^2 \tag{27}$$

This is a quadratic equation in α and can be solved analytically. We seek a real solution $0 \le \alpha \le 1$ for which $t_1(\alpha) \le b$. The same procedure is repeated for all contacts. Once all edge-crossings have been found and sorted, we have

[3] Another interesting property of this curve is that the solution given by trust-region methods lies on it.

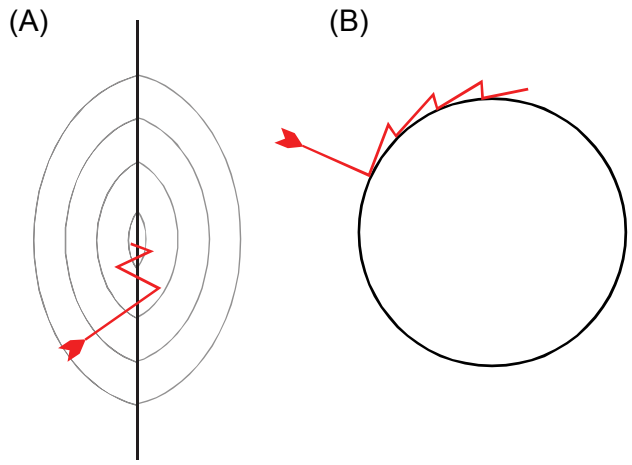

Fig. 4. Illustration of chattering phenomena caused by edges.

a set of sub-segments within which the objective function is smooth. We then apply a cubic linesearch within each sub-segment, and choose the best point we find as the next iterate.

D. Projection on straight and curved edges

We found that the edge-aware linesearch eliminated a substantial number of problematic cases, however another problem was now uncovered. Once $\mathbf{x}$ landed on an edge, the second-order convergence was often lost and instead the algorithm started to behave more like a first-order method, choosing the next iterate from the segment $\mathbf{x} : \mathbf{x}_\mathrm{C}$ instead of $\mathbf{x}_\mathrm{C} : \mathbf{x}_\mathrm{N}$. In retrospect the reason for this is obvious. The modified Jacobian (26) guarantees that $\mathbf{x}_\mathrm{C}$ lies within the tangent to the edge, but there is no such guarantee for $\mathbf{x}_\mathrm{N}$. As a result the segment $\mathbf{x}_\mathrm{C} : \mathbf{x}_\mathrm{N}$ deviates from the edge, the objective function increases along that segment, and so the algorithm effectively uses only the segment $\mathbf{x} : \mathbf{x}_\mathrm{C}$.

The solution to this problem is to make sure that both $\mathbf{x}_\mathrm{C}$ and $\mathbf{x}_\mathrm{N}$ lie in the tangent to the edge. This is done by forming the $(3n_c)$-by-$(3n_c - 1)$ matrix $B(\mathbf{x})$ whose columns span the $3n_c - 1$ dimensional tangent space. Then we parameterize the deviations from $\mathbf{x}$ within the tangent space as $\boldsymbol{\epsilon} = B\boldsymbol{\xi}$ and write the local quadratic model (23) in terms of $\boldsymbol{\xi}$. Finding the Cauchy and Newton points with respect to this restricted model and mapping back to the original space, we have

$$\begin{aligned} \mathbf{x}_\mathrm{C} &= \mathbf{x} - BB^\mathsf{T}\mathbf{g}\frac{\mathbf{g}^\mathsf{T}BB^\mathsf{T}\mathbf{g}}{\mathbf{g}^\mathsf{T}BB^\mathsf{T}HBB^\mathsf{T}\mathbf{g}} \\ \mathbf{x}_\mathrm{N} &= \mathbf{x} - B\left(B^\mathsf{T}HB\right)^{-1}B^\mathsf{T}\mathbf{g} \end{aligned} \tag{28}$$

This looks complicated but is in fact very efficient computationally. Indeed B is a block-diagonal matrix having one block per contact. For contacts which are not on an edge or where the objective function is decreasing away from the edge (i.e. the edge is inactive), the block is $I_{3\mathrm{x}3}$. For contacts on active edges, the blocks are given by

$$B_\mathrm{plane} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \; B_\mathrm{cone} = \begin{bmatrix} \mu(x_1 - b) & 0 \\ x_2 & -x_3 \\ x_3 & x_2 \end{bmatrix} \tag{29}$$

Note that if we replace B with RB in (28), where R is any orthonormal matrix, the resulting $\mathbf{x}_\mathrm{C}$ and $\mathbf{x}_\mathrm{N}$ remain unchanged. Thus the algorithm does not depend on the parameterization of the tangent space but only on the space itself.

This extension to the algorithm fixed all problems with planar edges. However we still observed chattering at conic edges, illustrated in **Fig 4B**. Our edge-aware linesearch lands on a conic edge, then the projection due to B causes a move in the tangent space as expected. Since the edge manifold is curved, this immediately takes us a (short) distance away from it. Then the edge-aware linesearch sends us back to the edge, and so on. The inherent problem is that the descent "direction" should actually be a curved surface, while we are using methods designed to work with linear subspaces. The obvious fix is to project the points $\mathbf{x}_\mathrm{C}$ and $\mathbf{x}_\mathrm{N}$ on the edge surface. This would be hard to do for a general surface, but since we are working with cones, the projection here can be done analytically. Focusing again on a single contact, we seek the point $\mathbf{t}$ on the friction cone which is closest to a given point $\mathbf{x}$. This constrained optimization problem can be handled using Lagrange multipliers. After some tedious algebra which we omit, the problem reduces to finding the roots of a certain quadratic polynomial. Thus $\mathbf{x}_\mathrm{C}$ and $\mathbf{x}_\mathrm{N}$ are projected on the friction cone analytically.

E. Pseudocode of the algorithm

repeat $MaxIt$ **times**
  if $\|\mathbf{r}(\mathbf{x})\|$ **is sufficiently small**
    return `"success"`
  find active edges and construct $B(\mathbf{x})$
  compute $\mathbf{x}_\mathrm{N}$**, project on active cone edges**
  if $\mathbf{x}_\mathrm{N}$ **is sufficiently better than** $\mathbf{x}$
    accept $\mathbf{x}_\mathrm{N}$
    decrease λ
    continue with next iteration
  increase λ
  compute $\mathbf{x}_\mathrm{C}$**, project on active cone edges**
  find edge crossings in $\mathbf{x} : \mathbf{x}_\mathrm{C}$ **and** $\mathbf{x}_\mathrm{C} : \mathbf{x}_\mathrm{N}$
  form list of sub-segments
  do cubic linesearch in each sub-segment
  if a better point is found
    accept best point found
    continue with next iteration
  else
    return `"local minimum"`
return `"max iterations exceeded"`

F. Initialization

The obvious way to initialize the algorithm in the context of a dynamic simulation is to use the solution from the previous time step (warm start). An alternative is to assume that all contacts are in the sticking state – which is true most of the time. Under this assumption the velocity $\mathbf{v}$ is known. Then we find $\mathbf{f} = A^\dagger(\mathbf{v} - \mathbf{v}_0)$ where $A^\dagger$ is the Moore-Penrose pseudoinverse, and set $\mathbf{x} = \mathbf{v} - \mathbf{f}$.

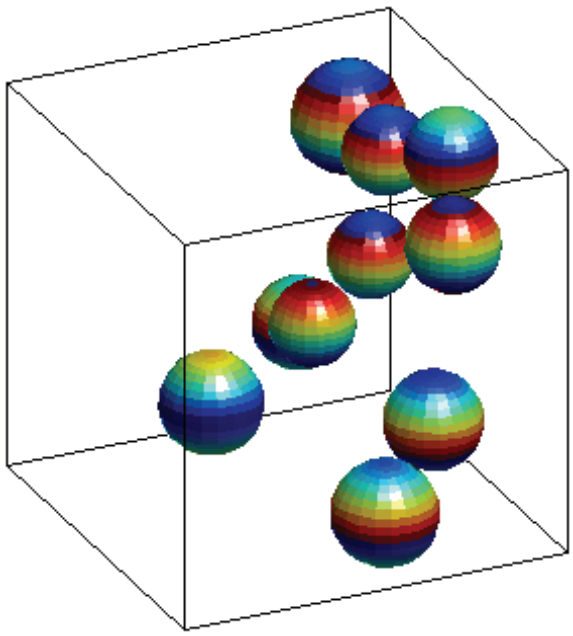

Fig. 5. One frame from our simulator.

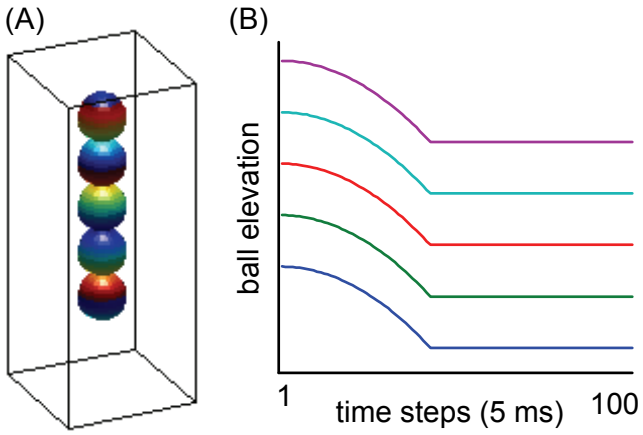

Fig. 6. Ball-drop test for shock propagation.

IV. Simulation Results

The different versions of the algorithm were tested on randomly generated problems as well as on problems arising within a dynamic simulation. Here we focus on the latter results. The simulator was written in Matlab specifically for testing the algorithm. Therefore we decided to keep everything other than the contact problem as simple as possible. To simplify collision detection, the simulation included n_b balls of different radii moving inside a cube. The balls were initialized at random positions and velocities and then their motion was simulated in the presence of contacts and gravity. We set $b = 0$. All results reported here were obtained with the initialization procedure which assumes sticking contacts (the results with warm starts were even better). When the algorithm terminated we accepted the result from the last iteration and proceeded with the next time step.

Fig 5 shows the first frame of one simulation with $n_b = 10$. A typical simulation run is shown in the movie which accompanies the paper. Each frame in the movie shows the number of contacts and the number of iterations of the algorithm. The time step in the movie is $h = 0.01$ sec. In other simulations we have used $h = 0.005$ sec.

Fig 6 shows the results of a simulation where 5 balls were initialized at vertically stacked positions (**Fig 6A**) and zero velocities. The balls then dropped together due to gravity, and the bottom ball eventually hit the floor. At this point the shock was propagated through the system within a single time step, and the vertical velocities of all balls became zero. The ball positions over time are shown in **Fig 6B**. Somewhat surprisingly this turned out to be a very easy problem – the algorithm only took one iteration to solve it.

In order to explore the behavior of the algorithm more systematically, we tested it 10 times in each of 12 conditions which differed by the friction coefficient $\mu = \{0.1,\ 0.2,\ 1.0,\ 2.0\}$ and by the number of balls $n_b = \{5,\ 10,\ 15\}$. Each test run continued for 200 time steps with $h = 0.005$ sec. The results are summarized below.

	$n_b = 5$ $n_c = 7$	$n_b = 10$ $n_c = 16$	$n_b = 15$ $n_c = 27$
$\mu = 0.1$	3.8 99 %	4.8 98 %	6.5 90 %
$\mu = 0.5$	2.6 95 %	5.3 85 %	7.5 73 %
$\mu = 1.0$	2.8 90 %	4.9 71 %	10.2 60 %
$\mu = 2.0$	2.9 88 %	4.6 71 %	16.2 55 %

(Table 1)

Each column is labeled with the number of balls n_b as well as the corresponding average number of contacts n_c measured in the simulation. The top row in each cell is the number of iterations per timestep. This is computed by taking the median in each simulation run, and then averaging over the 10 simulation runs within each condition. The bottom row is the percent iterations on which the Newton point $\mathbf{x}_\mathrm{N}$ was accepted. Recall that on those iterations the edge-aware linesearch is not used. On the remaining iterations the solution was almost always in the segment $\mathbf{x}_\mathrm{C} : \mathbf{x}_\mathrm{N}$. About half of these solutions were on edges, and the other half in smooth sub-segments between edges.

We find these results encouraging for several reasons. First, unlike some algorithms which have difficulties with large friction coefficients, we can handle very large friction. Indeed $\mu = 1$ is about as high as one would ever need in a physical simulation, while our algorithm works well with $\mu = 2$. **Table 1** shows that, for medium numbers of contacts, increasing μ does not actually make the problem harder for our algorithm. Second, the algorithm accepts $\mathbf{x}_\mathrm{N}$ on the large majority of iterations, which means that most of the time we are in the second-order convergence regime.

The small iteration count is particularly impressive in light of the fact that we are not using warm starts. Also, recall that one iteration of our algorithm involves the solution of a linear system which is about 3 times smaller than the corresponding system obtained with the LCP approach. Thus the algorithm not only uses few iterations, but each iteration is substantially faster compared to other methods. The additional mechanisms we introduced are tedious to derive and implement, however they are based on analytical formulas and overall require less computation than finding the Newton point.

V. Discussion and future work

In this paper we formulated contacts dynamics as an implicit nonlinear complementarity problem, and reduced it to unconstrained optimization of a semi-smooth objective function. After overcoming a number of obstacles due to the lack of smoothness, we obtained a very efficient algorithm. In robotics applications such as legged locomotion or hand manipulation, the cell corresponding to $\mu = 1$ and $n_c = 16$ in **Table 1** is likely to be typical. If our algorithm can indeed simulate robotic systems in 5 iterations and spend 70% of its time in the second-order convergence regime, it will be more efficient than any other known algorithm.

The simulation results presented here are based on a specific dynamical system. We do not think that the contact resolution problem for this system is substantially simpler than any other system with the same μ and n_c, but nevertheless more extensive simulations are needed. We are in the process of developing a general simulator for multi-joint dynamics with friction, which will allow us to study a wide range of mechanical systems. We plan to include both the present algorithm and the best LCP-based algorithms in the new simulator, and perform side-by-side comparisons. While this future work is important, a lot has already been accomplished: we developed an original approach to an important problem and presented encouraging numerical results suggesting that the new approach may be better than the state-of-the-art.

A. Possible extensions to the algorithm

Apart from testing the algorithm in the context of a general-purpose simulator, we are considering several extensions. The first is a mechanism for recovery from local minima. In the context of root-finding via minimization, a local minimum is a point $\mathbf{x}$ where $\mathbf{r}(\mathbf{x}) \neq 0$ while $J(\mathbf{x})^{\mathsf{T}} \mathbf{r}(\mathbf{x}) = 0$. This can only occur when $J(\mathbf{x})$ is singular – which is very rare. If a local minimum is encountered we could either restart the minimization at a different point, or perturb the equation as $D\mathbf{r}(\mathbf{x}) = 0$ where D is some diagonal matrix with positive elements on the diagonal. Such rescaling of the residual may also be useful for the purposes of preconditioning. For example, we can compute a suitable scaling matrix D based on the row-sums of the inverse inertia matrix A.

The handling of cone edges could also be improved. Currently we project $\mathbf{x}_{\mathrm{N}}$ and $\mathbf{x}_{\mathrm{C}}$ on the cone and proceed with linesearch. Instead of this Cartesian projection we could represent the local model (23) as a quadratic over a cone, correct for the curvature by including a term dependent on the derivative of $B(\mathbf{x})$, and move along the geodesics of the cone. These geodesics can be constructed analytically.

Finally, we are yet to analyze the energy of the simulated system and do careful stability tests. Our observations so far indicate that the simulation is surprisingly stable. This was the case even with earlier versions of the algorithm which resulted in chattering and often reached the maximum number of iterations before finding the global minimum. Yet we did not observe instability or other non-physical behavior. Unlike Lemke's algorithm which is designed to avoid cycling instead of reduce a sensible cost function, our algorithm improves the solution on every iteration. This may be the reason why it yields sensible results even when it is interrupted before the global minimum is found.

Acknowledgements

This work was supported by the National Science Foundation and the National Institutes of Health.

References

[1] D. Stewart and J. Trinkle. An implicit time-stepping scheme for rigid-body dynamics with inelastic collisions and Coulomb friction. *International Journal Numerical Methods Engineering* **39**: 2673–2691 (1996).

[2] D. Kaufman, S. Sueda, D. James and D. Pai. Staggered projections for frictional contact in multibody systems. *ACM Transactions on Graphics (SIGGRAPH ASIA)*, **164**: 1–11 (2008).

[3] M. Anitescu and G. Hart. A fixed-point iteration approach for multibody dynamics with contact and small friction. *Mathematical Programming* **101**: 3–32 (2004).

[4] K. Yamane and Y. Nakamura. A numerically robust LCP solver for simulating articulated rigid bodies in contact. *Proceedings of Robotics: Science and Systems* **4** (2008).

[5] J. Lloyd. Fast implementation of Lemke's algorithm for rigid body contact simulation. *Proceedings of the IEEE International Conference on Robotics and Automation*, 4538–4543 (2005).

[6] F. Pfeiffer and C. Glocker. *Multibody dynamics with unilateral constraints*. Wiley Series in Nonlinear Science (2006).

[7] S. Dirkse and M. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods Software* **5**: 123–156 (1995).

[8] M. Ferris, C. Kanzow and T. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming* **86**: 475–497 (1999).

[9] D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research* **19**: 352–389 (1994).

[10] H. Jiang. Global convergence analysis of the generalized Newton and Gauss-Newton methods of the Fischer-Burmeister function for the complementarity problem. *Mathematics of Operations Research* **24**: 529–543 (1999).

[11] L. Qi and J. Sun. A nonsmooth version of Newton's method. *Mathematical Programming* **58**: 353–367 (1993).