Implementation of Multi-rigid-body Dynamics within a Robotic Grasping Simulator

Andrew T. Miller Henrik I. Christensen
Centre for Autonomous Systems
Royal Institute of Technology
Stockholm, Sweden
amiller, hic@nada.kth.se

Abstract

Robotic simulation systems allow researchers, engineers, and students to test control algorithms in a safe environment, but until recently these systems only simulated the dynamics of the mechanism itself and could not simulate (complex) contacts with other bodies in the environment. However, if the robot's task involves grasping an object, accurate simulation of contact and friction forces is a necessity. Recently developed methods formulate the constrains as a linear complementarity problem, allowing a solution to be computed using proven algorithms, but for anyone implementing such a system, several additional considerations must be taken into account. In this paper we present the implementation of the dynamics module of our freely available grasping simulator and present an example grasping task.

1 Introduction

As robotic simulation systems become more accurate at modeling the real world, the number of possible applications for these systems increases. In commercial systems simulation is typically a necessity to allow testing of different operations before a product is actually manufactured or a new production line is setup. Thus every robot manipulator manufacturer provides a test suite with their robots. One example of this the ABB Robot Studio in which basic assembly, welding, etc. can be evaluated.

Simulation has also always been useful in teaching robotics [3, 11, 10]. These packages allow students to write programs to control a robot, that if faulty will not result in costly damage to a real robot. For a robotics researcher they provide a test bed for experimentation with different control algorithms. In addition, for some tasks such as learning, the number of iterations needed to implement a control algorithm makes it impossible to perform the learning on-line, and the researcher is thus required to use simulation systems to run say 1000 or 1 million generations of a system. However, many current systems only simulate the dynamics of the robot and cannot simulate any interactions with its environment. The key reason for this is the difficulty of efficiently and accurately modeling frictional contacts. Recent work, however, has opened the door for a wider variety of more complex simulation systems.

One branch of robotics that can clearly benefit is grasping. At its core robotic grasping involves the forming and breaking of contacts between the links of a robotic hand and one or more objects in the environment. Like other simulation systems, a grasping simulator allows a researcher to evaluate different control algorithms, but it can also serve as a planning environment for the larger grasping task, including reach planning, grasp selection, and object acquisition. Working toward this goal, we have created a system, known as "GraspIt!"[1] [7], that can import a wide variety of robot and object models, evaluate grasps formed between a hand and an object, and allow a user to visualize results of the analysis. Recently, we have implemented a dynamics module that computes the motions of a group of connected robot elements, such as an arm and a hand, under the influence of controlled motor forces, joint constraint forces, contact forces and external forces. This allows a user to dynamically simulate an entire grasping task.

The purpose of this paper is to describe our dynamic simulation method, which is based on the theoretical work of Anitescu and Potra [1]. The paper focuses in particular on the integration of the theoretical framework into an operational system and the required adaptations to allow operation in a real environment.

In section 2, we review the formulation of contacts and joints as linear equality and inequality constraints. Next in

[1] This system will soon be available for download for a variety of platforms from http://www.cs.columbia.edu/~amiller/graspit.

section 3 we present an overview the simulator, so that we can discuss our algorithm for advancing the simulation by one time step in section 4. Section 5 discusses our current methods of joint control and section 6 presents an example grasping task. Lastly, section 7 presents our plans for extending the system.

2 Dynamic Simulation

A collision between two objects takes place over a finite period of time and even in the case of rigid bodies, it involves a complicated surface deformation before the objects separate again. One of the most accurate ways of modeling this interaction is with finite element methods (FEMs), but these techniques are computationally challenging [5]. The most recent work either uses an impulse based method or a linear constraints method. In the impulse based approach [8, 9], a pair of instantaneous equal and opposite impulses are applied to the two contacting bodies which immediately changes their velocities and prevents inter-penetration from occurring. These methods are very good at modeling systems like a vibrating parts feeder but have difficulty handling resting contacts. In the linear constraints method [2, 12, 1], the contact forces are solved for analytically. The non-penetration constraints and frictional forces are expressed as inequalities that can be used to formulate the problem as a linear complementarity problem (LCP). This type of problem can then be solved with Lemke's algorithm, which is a pivoting method similar to the simplex algorithm for linear programming [4]. Because grasping often involves sustained contacts, the latter method is the most appropriate. We chose to use the method described by Anitescu and Potra [1] because it explicitly includes joint constraints, but an actual implementation required several modifications.

We start by defining $\boldsymbol{v}(t) \in \Re^{6\times n}$ as the generalized velocity vector of a system of n rigid bodies at time t. The Newtonian equations for such a system that includes multiple joined bodies and multiple unilateral contacts can be written as follows:

$$\boldsymbol{M}\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}_j + \boldsymbol{f}_l + \boldsymbol{f}_n + \boldsymbol{f}_f + \boldsymbol{k}, \qquad (1)$$

where $\boldsymbol{M}$ is the positive definite, symmetric mass-inertia matrix, and the right side of the equation is the sum of generalized forces acting on the bodies: $\boldsymbol{f}_j$ is the vector of joint constraint forces, $\boldsymbol{f}_l$ is the vector of joint limit forces, $\boldsymbol{f}_n$ is the vector of contact normal forces, $\boldsymbol{f}_f$ is the vector of contact friction forces, and $\boldsymbol{k}$ is the vector of Coriolis, centripetal, gravitational, and other external forces. If we wish to find the velocity of the bodies at discrete instants in time, we can use an Euler integration scheme:

$$\boldsymbol{M}(\boldsymbol{v}^{l+1} - \boldsymbol{v}^l) = h(\boldsymbol{f}_j^l + \boldsymbol{f}_l^l + \boldsymbol{f}_n^l + \boldsymbol{f}_f^l + \boldsymbol{k}^l), \qquad (2)$$

where the superscript l denotes time step number and h is the length of the time step. However, on the right side of this equation, only the external forces are known, and we must solve for the contact and joint forces in addition to the new velocity. This is done by using several equality and inequality constraints which are explained in the following paragraphs.

2.1 Contact Constraints

To handle contacts, we examine a system of two bodies with a single contact between them, and then later generalize to multiple contacts when we describe our algorithm in section 4. We define c_n as the magnitude of the contact normal impulse. This is simply the magnitude of the contact force multiplied by the length of the time step, or $h\|\boldsymbol{f}_n^l\|$. We then let $\boldsymbol{n}c_n$ be the 12×1 vector of two opposing generalized normal impulses exerted on the two bodies. Since contacts can only apply compressive forces, we have the constraint $c_n \geq 0$, and since the bodies cannot be allowed to interpenetrate, we have the constraint that the relative velocity of the bodies at the contact points must be greater or equal to zero ($\boldsymbol{n}^T\boldsymbol{v}^{l+1} \geq 0$). These two constraints are complementary since *either* a force is being applied at a contact and the relative velocity of the bodies at the contact is 0, *or* the bodies are separating and no force can be applied. This constraint can be written as a complementarity condition in the following way:

$$\boldsymbol{n}^T\boldsymbol{v}^{l+1} \geq 0 \quad \text{compl. to} \quad c_n \geq 0. \qquad (3)$$

In addition to the contact normal force, there is also a certain amount of friction acting at any contact. We assume that the magnitude of the frictional force is bounded by a convex limit surface that contains the origin, and this limit surface is scaled by the size of the contact normal force. If there is no relative motion at the contact, then the frictional force must lie on or within the boundary of the scaled limit surface, but if there is a relative velocity between the two bodies at the contact, then the frictional force must maximize power dissipation and therefore lie on the boundary of the limit surface. In the simple case of Coulomb friction, frictional forces are limited to lie within a circle in the plane perpendicular to the contact normal. The radius of this circle is equal to the coefficient of friction, μ, times the magnitude of the contact normal force. This means that the total contact force must lie within a cone, commonly known as a friction cone. To express this constraint linearly, we approximate the cone with a convex

polyhedron $\mathcal{F}$:

$$\mathcal{F} = \left\{ \boldsymbol{n}c_n + \boldsymbol{D}\boldsymbol{\beta} \mid c_n \geq 0,\ \boldsymbol{\beta} \geq 0,\ \boldsymbol{e}^T\boldsymbol{\beta} \leq \mu c_n \right\}, \quad (4)$$

where $\boldsymbol{e} = [1, 1, \ldots, 1]^T \in \Re^u$ with u being the number of edges in the polyhedral approximation, $\boldsymbol{\beta} \in \Re^u$ is a vector of weights, and the columns of $\boldsymbol{D}$ are direction vectors that positively span the space of possible generalized friction forces. We assume if a vector $\boldsymbol{d}$ is one of the columns of $\boldsymbol{D}$, then so is $-\boldsymbol{d}$. Although we currently are only simulating friction in the tangent plane of the contact, with this general friction formulation we can model anisotropic Coulomb friction and torsional friction simply by modifying the columns of $\boldsymbol{D}$.

This formulation of the friction constraint leads to two complementarity conditions:

$$\boldsymbol{e}\lambda + \boldsymbol{D}^T\boldsymbol{v}^{l+1} \geq 0 \quad \text{compl. to} \quad \boldsymbol{\beta} \geq 0, \quad (5)$$

$$\mu c_n - \boldsymbol{e}^T\boldsymbol{\beta} \geq 0 \quad \text{compl. to} \quad \lambda \geq 0. \quad (6)$$

If $\mu c_n - \boldsymbol{e}^T\boldsymbol{\beta} > 0$, meaning the friction force is strictly interior to the friction cone, then $\lambda = 0$ and $\boldsymbol{D}^T\boldsymbol{v}^{l+1} \geq 0$. But for any column $\boldsymbol{d}$ of $\boldsymbol{D}$ such that $\boldsymbol{d}^T\boldsymbol{v}^{l+1} > 0$ then there exists another column corresponding to $-\boldsymbol{d}$ and $-\boldsymbol{d}^T\boldsymbol{v}^{l+1} < 0$ thus contradicting the non-negativity of $\boldsymbol{D}^T\boldsymbol{v}^{l+1}$. So $\boldsymbol{D}^T\boldsymbol{v}^{l+1} = 0$, which means there is no relative motion in the friction directions at the contact. On the other hand, if there is relative motion at the contact then $\lambda > 0$ and $\mu c_n - \boldsymbol{e}^T\boldsymbol{\beta} = 0$, meaning the contact force lies on the boundary of the friction cone, and the direction of the friction force will oppose the relative motion.

2.2 Joint Constraints

A joint constraint can be expressed as equality constraints on the relative velocities of the two connected bodies as follows:

$$\boldsymbol{J}^T\boldsymbol{v}^{l+1} = 0. \quad (7)$$

A prismatic or a revolute joint will constrain 5 of the 6 relative velocities between the two connected bodies with respect to the joint coordinate frame on each body. Each joint thus adds 5 rows to the matrix $\boldsymbol{J}^T$. Passively coupled joints add a 6th row that constrains the joint's velocity relative to the velocity of the joint it is coupled to. Fixed joints, where one robot is attached to another, constrain all relative velocities and add 6 rows. For each constraint we have one unknown constraint force, but we denote it as an impulse:

$$h\boldsymbol{f}_j^l = \boldsymbol{J}\boldsymbol{c}_j, \quad (8)$$

where $\boldsymbol{c}_j$ is the vector of magnitudes of the joint constraint impulses.

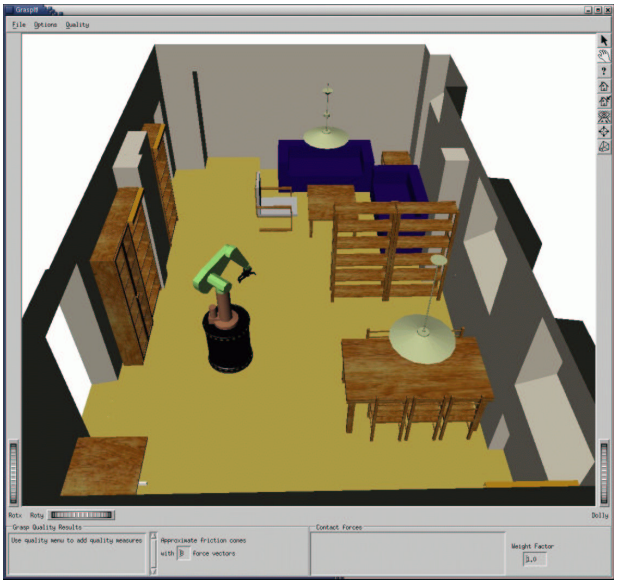


Figure 1: A robot or robotic platform can operate within a user defined world. In this case it is the manipulation platform and living room environment at the Center for Autonomous Systems.

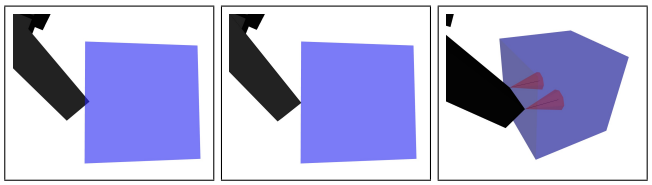
If any joint is at its minimum or maximum value, we can formulate another inequality constraint similar to the contact normal constraint in equation 3:

$$\boldsymbol{l}^T\boldsymbol{v}^{l+1} \geq 0 \quad \text{compl. to} \quad c_l \geq 0, \quad (9)$$

where $\boldsymbol{l}c_l$ is the impulse necessary to prevent the joint from going past its limit.

3 GraspIt! Overview

The framework described above was incorporated into our grasping simulator, GraspIt!. The system consists of several modules, including one to load and construct a variety of object models and robot designs, a collision detection and contact determination system, a grasp analysis suite, visualization methods to see the results of the analysis, and the dynamics module. The focus of this paper is on the dynamics system, but information about the other components can be found in an another article [7]. Before discussing our implementation of the dynamics, we first describe our definitions of the various body types used in the simulation and then give an overview of our collision detection and contact determination scheme which is responsible for finding contacts between bodies during each time step of the dynamics.



(a) (b) (c)

Figure 2: The collision detection and contact location process: (a) The collision of a link of the Barrett hand with the side of a cube is detected. (b) A search is conducted to find the instant when the link is within 0.1mm of the surface. (c) The geometry of the contact is determined and friction cones are placed at the vertices bounding the contact region (in this case a line).

3.1 Body Types

A basic body consists of a pointer to its geometry, a material specification, a list of contacts, and a transform that specifies the body's pose relative to the world coordinate system. The body geometry is defined in a scene graph, similar to VRML 1.0. The material is one of a set of predefined material types and is used when computing the coefficient of friction between two contacting bodies. A dynamic body inherits all of the properties of a body and defines the mass of the body, the location of its center of mass relative to the body frame, and its inertia tensor. It also includes the body's dynamic state parameters, $\boldsymbol{q}$ and $\boldsymbol{v}$, which specify the pose and velocity of a body frame located at the center of mass relative to the world coordinate system.

The reason for distinguishing between bodies and dynamic bodies is that some bodies are simply considered obstacles, and while they are elements of the collision detection system and can provide contacts on other bodies, they are not part of the dynamics computations and remain static. This makes it possible to create a complex world full of obstacles without making the dynamics intractable to compute (an example of such a world is shown in figure 1).

3.2 Collision Detection and Contact Determination

To prevent bodies from passing through each other during the simulation or while they are being manipulated by the user in a static setting, the system performs rapid collision detection using the PQP system (Proximity Query Package) [6]. When a body is loaded into the simulator, it is faceted by the renderer, and its collection of triangles is passed to PQP where they become the leaves of a hierarchical tree of bounding volumes. Fast recursive algorithms can then determine if any of the triangles of one body intersect any of the triangles of another body, or can provide a minimum distance between the two bodies.

If a collision is detected (see figure 2(a)), the motion of the bodies must be reversed back to the point when the contact first occurs. To find this instant, GraspIt! begins by moving the objects to their previous locations before the collision and calls the PQP system to determine the minimum distance between them. Since body transforms must be represented with fixed precision floating point numbers, it is impossible to find the instant of exact contact between two bodies. Thus, we define a thin contact region around the surface of each body, and if the distance between two bodies is less than this threshold (currently set at 0.1mm) then they are considered to be in contact. We use a binary search technique to move two bodies to within this distance after they have collided (see figure 2(b)).

After the bodies have been moved to within the contact distance, the system must determine the regions of contact between the two bodies. This is done using a modification of the PQP distance query algorithm that computes the overlap between all pairs of triangles that are within the contact distance. It performs topological consistency checks to ensure that the contacts are valid, and in the case of a planar contact, it finds the planar convex hull of the contact set and removes any interior contact points. These interior contact points do not affect the mechanics of the grasp, because for any contact with a distribution of forces along a line or an area, the wrench applied at the contact can be represented as a single resultant wrench. This can then be specified as a convex sum of forces acting at points on the boundary of the contact region. Once the final set of contacts between the two bodies has been determined, the system draws a red friction cone at each contact point, which serves to visually mark the position of the contact and its normal (see figure 2(c)).

4 One Time Step

After the robots and bodies have been loaded into the world environment, the user can start and stop the dynamic simulation at any time. To advance the simulation from time step l to time step $l + 1$, the system performs the following steps:

1. Because the pose of each body is represented using Euler parameters to avoid the singularities present in other representations, we must define a matrix $\boldsymbol{G}$ that will relate the 6×1 velocity vector of each body to the 7×1 vector of the time derivatives of its pose.

So for each dynamic body, we start by building the matrix $\boldsymbol{B}$, which relates the angular velocity of body i (expressed with respect to the body's coordinate frame) to the time derivatives its Euler parameters:

$$\dot{\boldsymbol{q}}_{rot} = \boldsymbol{B}\,{}^b\boldsymbol{\omega}$$
$$\begin{bmatrix} \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \\ \dot{q}_7 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -q_5 & -q_6 & -q_7 \\ q_4 & -q_7 & q_6 \\ q_7 & q_4 & -q_5 \\ -q_6 & q_5 & q_4 \end{bmatrix}\begin{bmatrix} {}^b\omega_x \\ {}^b\omega_y \\ {}^b\omega_z \end{bmatrix}, \tag{10}$$

where $q_4 \ldots q_7$ are the values of the Euler parameters of body i at time step l. Using this conversion matrix, we build a 7x6 matrix, $\boldsymbol{G}$, that converts a body's velocity, expressed in world coordinates, to the time derivative of its position:

$$\begin{aligned} \dot{\boldsymbol{q}} &= \boldsymbol{G}\boldsymbol{v} \\ \dot{\boldsymbol{q}} &= \begin{bmatrix} \boldsymbol{I}_3 & 0 \\ 0 & \boldsymbol{B}\boldsymbol{R}^{\boldsymbol{T}} \end{bmatrix}\boldsymbol{v}, \end{aligned} \tag{11}$$

where $\boldsymbol{R}$ is the current orientation of the body expressed as a 3x3 rotation matrix. Combining the $\boldsymbol{G}$ matrix for each body in block diagonal form, we create the matrix $\tilde{\boldsymbol{G}}$.

2. Move all bodies along their current trajectories using the time step h (our default value is 2.5 milliseconds) using the equation

$$\boldsymbol{q}^{l+1} = \boldsymbol{q}^l + h\tilde{\boldsymbol{G}}\boldsymbol{v}^l. \tag{12}$$

3. Check for collisions and joint limits. If any bodies have interpenetrated, perform a binary search on the h value looking for the instant when the closest two bodies are separated by a distance between 0 and 0.1mm. If any joint limits were exceeded, perform the same search to find the instant when the joint is within 0.1 degrees from its limit for revolute joints and 0.1mm from its limit for prismatic joints. This smallest value of h from these two searches is used in the last step.

4. Find all contacts given the current locations of the bodies.

5. Determine the coefficient of friction, μ, for each contact as follows:

$$\mu = \begin{cases} \mu_s, & \|\boldsymbol{\Delta v_c^l}\| < s_t \\ \mu_k, & \|\boldsymbol{\Delta v_c^l}\| \geq s_t \end{cases}, \tag{13}$$

where μ_s and μ_k are the coefficients of static and kinetic friction for the pair of contacting materials, $\boldsymbol{\Delta v_c^l}$ is the relative velocity of the two contacting bodies at the contact point, and s_t is a threshold speed.

6. Compute the external forces, $\boldsymbol{k}$, acting on the bodies of the island. This consists of gravitational and inertial forces.

7. Apply joint control forces (described below). These are accumulated on the bodies connected by joints and added to $\boldsymbol{k}$.

8. Apply joint friction forces. This is computed using the following equation:

$$\tau_{friction} = c\ sgn(\dot{\theta}) + v\dot{\theta}, \tag{14}$$

where c represents constant Coulomb friction that only depends on the sign of the joint velocity, and v is the coefficient of viscous friction which scales with the joint velocity. Like the control forces, these forces are accumulated in $\boldsymbol{k}$.

9. Divide the dynamic bodies into islands, where the bodies of an individual island are connected by either a joint or a contact.

10. For each island:

Compute the velocity of each body in the island, $\boldsymbol{v}^{l+1}$, using the methods described below.

If the bodies of a given island have no joints and no contacts, we can take a simple Euler step to find their next velocities:

$$\boldsymbol{v}^{l+1} = \boldsymbol{v}^l + h\boldsymbol{M}^{-1}\boldsymbol{k}. \tag{15}$$

If the bodies in an island have joints connecting them, we build the the joint constraint matrix, $\tilde{\boldsymbol{J}}^T$, that is the combined matrix of all of the joint constraints between the bodies of the island as described in equation 7. However, because we are solving for velocities and cannot constrain the relative position of the bodies, the bodies will tend to drift away from their joint connections over time due to small errors in the system. To correct this, we change the right hand side of equation 7 from 0 to a correction velocity, whose value is

$$\tilde{\boldsymbol{\epsilon}}_j = -\gamma\frac{\boldsymbol{\Delta p}}{h}, \tag{16}$$

where $\boldsymbol{\Delta p}$ is a vector of positional errors for all of the constraints, and γ is an error correction parameter that has a value between 0 and 1 and controls how fast the error is corrected. A value of 1 will add a velocity large enough to correct the error in the next time step but will cause overshoot in the next, so we currently use a value of 0.2.

If the bodies of a given island have only joint constraints, and have no contacts or joint limit constraints, we can solve the following linear system:

$$\begin{bmatrix} \boldsymbol{M} & -\tilde{\boldsymbol{J}} \\ \tilde{\boldsymbol{J}}^T & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{v}^{l+1} \\ \tilde{\boldsymbol{c}}_j \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}\boldsymbol{v}^l + h\boldsymbol{k} \\ \tilde{\boldsymbol{\epsilon}}_j \end{bmatrix}, \tag{17}$$

to find the velocity of the bodies, $\boldsymbol{v}^{l+1}$, and the magnitude of the joint constraint impulses $\tilde{\boldsymbol{c}}_j$.

If the bodies in an island have at least one contact or one joint limit reached, we must formulate a mixed LCP similar to the one described in [1]:

$$\begin{bmatrix} \boldsymbol{M} & -\tilde{\boldsymbol{J}} & -\tilde{\boldsymbol{l}} & -\tilde{\boldsymbol{n}} & -\tilde{\boldsymbol{D}} & 0 \\ \tilde{\boldsymbol{J}}^T & 0 & 0 & 0 & 0 & 0 \\ \tilde{\boldsymbol{l}}^T & 0 & 0 & 0 & 0 & 0 \\ \tilde{\boldsymbol{n}}^T & 0 & 0 & 0 & 0 & 0 \\ \tilde{\boldsymbol{D}}^T & 0 & 0 & 0 & 0 & \tilde{\boldsymbol{E}} \\ 0 & 0 & 0 & \tilde{\mu} & -\tilde{\boldsymbol{E}}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^{l+1} \\ \tilde{\boldsymbol{c}}_j \\ \tilde{\boldsymbol{c}}_l \\ \tilde{\boldsymbol{c}}_n \\ \tilde{\boldsymbol{\beta}} \\ \tilde{\boldsymbol{\lambda}} \end{bmatrix} + \boldsymbol{b} = \begin{bmatrix} 0 \\ 0 \\ \tilde{\boldsymbol{\xi}} \\ \tilde{\boldsymbol{\rho}} \\ \tilde{\boldsymbol{\sigma}} \\ \tilde{\boldsymbol{\zeta}} \end{bmatrix}$$

$$\begin{bmatrix} \tilde{\boldsymbol{c}}_l \\ \tilde{\boldsymbol{c}}_n \\ \tilde{\boldsymbol{\beta}} \\ \tilde{\boldsymbol{\lambda}} \end{bmatrix}^T \begin{bmatrix} \tilde{\boldsymbol{\xi}} \\ \tilde{\boldsymbol{\rho}} \\ \tilde{\boldsymbol{\sigma}} \\ \tilde{\boldsymbol{\zeta}} \end{bmatrix} = 0, \begin{bmatrix} \tilde{\boldsymbol{c}}_l \\ \tilde{\boldsymbol{c}}_n \\ \tilde{\boldsymbol{\beta}} \\ \tilde{\boldsymbol{\lambda}} \end{bmatrix} \geq 0, \begin{bmatrix} \tilde{\boldsymbol{\xi}} \\ \tilde{\boldsymbol{\rho}} \\ \tilde{\boldsymbol{\sigma}} \\ \tilde{\boldsymbol{\zeta}} \end{bmatrix} \geq 0. \tag{18}$$

In this equation, if we have m joints, p reached joint limits, and q contacts in the island, the constraints and variables are combined into the following matrices:

$$\begin{aligned} \tilde{\boldsymbol{J}} &= \left[\boldsymbol{J}^{(1)}, \ldots, \boldsymbol{J}^{(m)}\right], & \tilde{\boldsymbol{l}} &= \left[\boldsymbol{l}^{(1)}, \ldots, \boldsymbol{l}^{(p)}\right], \\ \tilde{\boldsymbol{n}} &= \left[\boldsymbol{n}^{(1)}, \ldots, \boldsymbol{n}^{(q)}\right], & \tilde{\boldsymbol{D}} &= \left[\boldsymbol{D}^{(1)}, \ldots, \boldsymbol{D}^{(q)}\right], \\ \tilde{\boldsymbol{E}} &= \text{diag}\left(\boldsymbol{e}^{(1)}, \ldots, \boldsymbol{e}^{(q)}\right), & \tilde{\boldsymbol{\mu}} &= \text{diag}\left(\mu^{(1)}, \ldots, \mu^{(q)}\right), \end{aligned}$$

and vectors:

$$\begin{aligned} \tilde{\boldsymbol{c}}_j &= \left[c_j^{(1)}, \ldots, c_j^{(m)}\right]^T, & \tilde{\boldsymbol{c}}_l &= \left[c_l^{(1)}, \ldots, c_l^{(p)}\right]^T, \\ \tilde{\boldsymbol{c}}_n &= \left[c_n^{(1)}, \ldots, c_n^{(q)}\right]^T, & \tilde{\boldsymbol{\beta}} &= \left[\boldsymbol{\beta}^{(1)T}, \ldots, \boldsymbol{\beta}^{(q)T}\right]^T, \\ \tilde{\boldsymbol{\lambda}} &= \left[\lambda^{(1)}, \ldots, \lambda^{(q)}\right]^T. \end{aligned}$$

We have modified the constant vector $\boldsymbol{b}$ from the one originally presented to add error correction velocities as follows:

$$\boldsymbol{b} = \left[\ -\boldsymbol{M}\boldsymbol{v}^l - h\boldsymbol{k}, \quad -\tilde{\boldsymbol{\epsilon}}_j, \quad 0, \quad -\tilde{\boldsymbol{\epsilon}}_n, \quad 0, \quad 0\ \right]^T,$$

where $\tilde{\boldsymbol{\epsilon}}_j$ is the error correction velocity for joint constraints already described, and $\tilde{\boldsymbol{\epsilon}}_n$ is an error correction velocity for contacts. Again, because we are constraining contact velocities rather than positions, error can build up and bodies can eventually interpenetrate. To prevent this we add the correction as follows:

$$\boldsymbol{\epsilon}_n^{(j)} = \begin{cases} \gamma\frac{\Delta p}{h}, & \Delta p < 0 \\ 0, & \Delta p \geq 0 \end{cases}, \tag{19}$$

where Δp is the difference between the current separation of two bodies at contact j and half the contact threshold distance (0.05mm). If the bodies move closer together than half the contact threshold distance, this velocity will push them back apart.

Next we need to convert the mixed LCP into a pure LCP by eliminating all of the equality constraints. To do this we rewrite equation 18 as follows:

$$\begin{bmatrix} \boldsymbol{M} & -\tilde{\boldsymbol{J}} & -\boldsymbol{H} \\ \tilde{\boldsymbol{J}}^T & 0 & 0 \\ \boldsymbol{H}^T & 0 & \boldsymbol{N} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^{l+1} \\ \tilde{\boldsymbol{c}}_j \\ \boldsymbol{z} \end{bmatrix} + \begin{bmatrix} -\boldsymbol{a} \\ -\tilde{\boldsymbol{\epsilon}}_j \\ \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{w} \end{bmatrix}$$

$$\boldsymbol{z} \geq 0,\ \boldsymbol{w} \geq 0,\ \boldsymbol{z}^T\boldsymbol{w} = 0. \tag{20}$$

Then we can eliminate the first two rows:

$$\begin{aligned} \boldsymbol{v}^{l+1} &= \boldsymbol{M}^{-1}(\tilde{\boldsymbol{J}}\tilde{\boldsymbol{c}}_j + \boldsymbol{H}\boldsymbol{z} + \boldsymbol{a}) \\ \tilde{\boldsymbol{c}}_j &= -\left(\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}\tilde{\boldsymbol{J}}\right)^{-1}\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}(\boldsymbol{H}\boldsymbol{z} + \boldsymbol{a}) + \\ &\quad \left(\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}\tilde{\boldsymbol{J}}\right)^{-1}\tilde{\boldsymbol{\epsilon}}_j, \end{aligned}$$

to arrive at the pure LCP:

$$\begin{aligned} (\boldsymbol{G} + \boldsymbol{N})\boldsymbol{z} + \boldsymbol{g} &= \boldsymbol{w} \\ \boldsymbol{z} \geq 0,\ \boldsymbol{w} \geq 0,\ \boldsymbol{z}^T\boldsymbol{w} &= 0, \end{aligned} \tag{21}$$

where

$$\begin{aligned} \boldsymbol{G} &= \boldsymbol{H}^T\boldsymbol{M}^{-1}\boldsymbol{H} - \\ &\quad \boldsymbol{H}^T\boldsymbol{M}^{-1}\left(\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}\tilde{\boldsymbol{J}}\right)^{-1}\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}\boldsymbol{H}, \\ \boldsymbol{g} &= \boldsymbol{b} + \boldsymbol{H}^T\boldsymbol{M}^{-1}\boldsymbol{a} - \\ &\quad \boldsymbol{H}^T\boldsymbol{M}^{-1}\left(\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}\tilde{\boldsymbol{J}}\right)^{-1}\left(\tilde{\boldsymbol{J}}^T\boldsymbol{M}^{-1}\boldsymbol{a} - \tilde{\boldsymbol{\epsilon}}_j\right). \end{aligned}$$

This LCP can be solved with Lemke's algorithm to find the value of $\boldsymbol{z}$, which can then be substituted back to find the value of $\tilde{\boldsymbol{c}}_j$ and ultimately $\boldsymbol{v}^{l+1}$.

5 Joint Control

Without the application of external motor forces on the links of a robot, they would fall limply under the influence of gravity. So in every iteration of the dynamics, a control routine is called for each robot. This routine can be written by the user, but we have provided some default routines. If given a new desired position for the end-effector of a robot, a trajectory generator creates a linear path in Cartesian space that has a continuous velocity and acceleration profile. The inverse kinematics of the robot are then computed for each sample along this path, and the joint trajectories are recorded. Or, if given a new set of joint positions, the trajectory generator creates the smooth joint trajectories individually.

We are currently using simple PD controllers to control the motor forces of each joint. Given a joint position set point from the trajectory, the controller for that joint determines the current error from that position and computes a

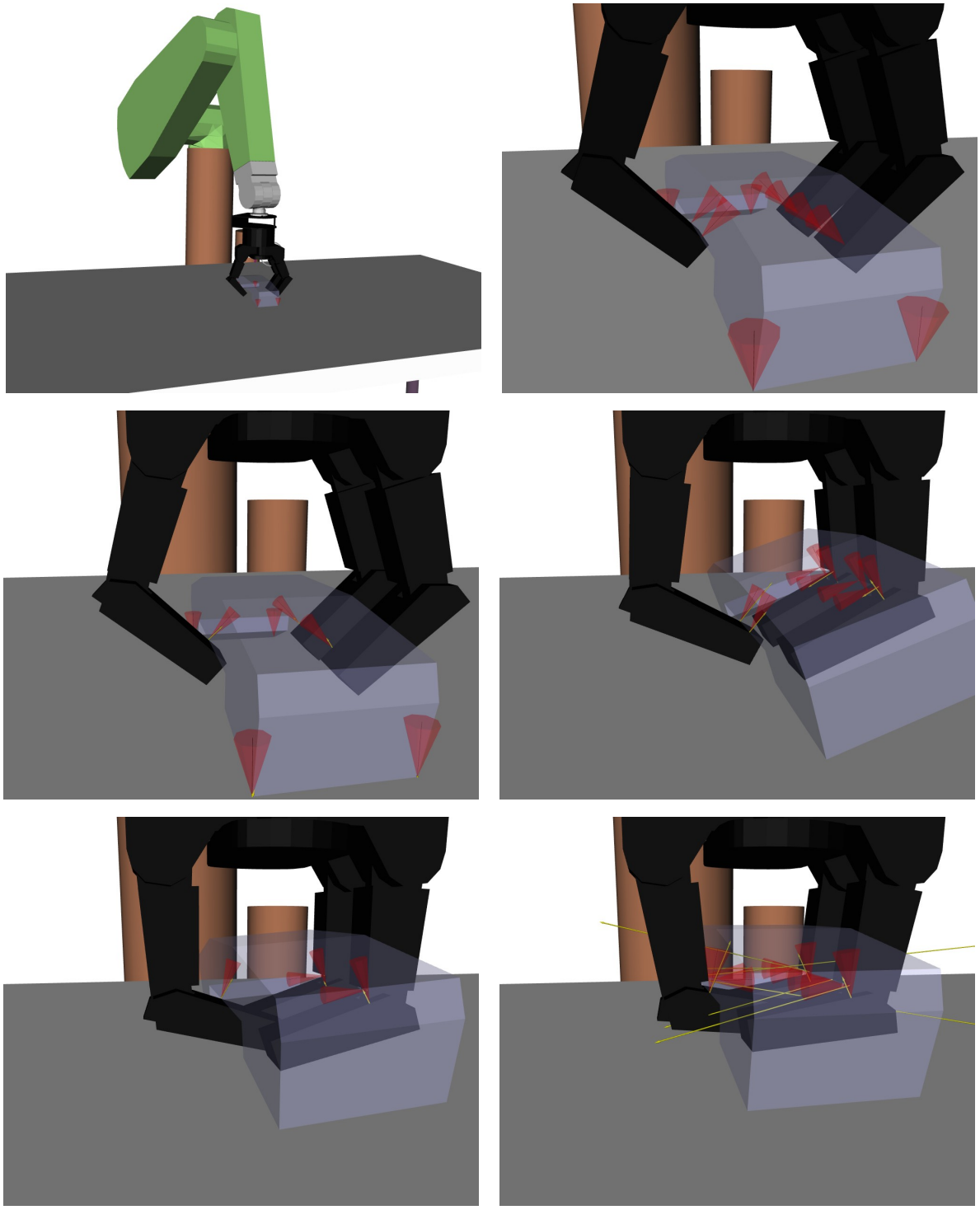


Figure 3: Top Left: Initial position of the Barrett hand and phone handset. Top Right: An auto-grasp of the static handset does not result in a force-closure grasp. Middle and Bottom Rows: An auto-grasp of the handset with dynamics turned on results in the handset being pulled into the hand, and the final grasp has force-closure. The computed contact forces are shown as arrows within the contact friction cones. The snapshots were taken at 1.2, 1.9, 2.2, and 2.7 seconds of simulation time.

joint force using gains defined within the robot configuration file. Once a joint force is computed, it is applied to the two connected bodies in opposite directions along or about the joint axis.

6 Simulating Grasp Formation

With the dynamics in place, it is possible to study the temporal formation of grasps. In this example, the Barrett hand, held by the Puma arm, is positioned above a phone handset which rests on the workbench. Leaving the object in a static state, an auto-grasp is performed where the 3 fingers are closed at the same speed until contact occurs. The result is shown at the top of figure 3. This is not a force-closure grasp because the fingers only contact the handset on its bottom surface. However, by turning on the dynamic simulation, and performing the auto-grasp again, the handset is pulled up into the hand until it reaches a force-closure state. For this test, the PD controllers of the 6 Puma joints simply attempt to maintain their positions in order to keep the wrist stable. The trajectory generator created trajectories for the three active finger joints that would completely close the fingers, but along their path the fingers contact and begin to lift the phone. The PD controllers of those joints then must apply additional torque to maintain the finger trajectory. Eventually the handset is lifted into the hand and the fingers can no longer move. In order to keep the grasp stable, the maximum joint torques for the two finger joints were each set to half the value of the thumb joint.

7 Future Directions

In this paper we have shown a method of simulating the dynamics of robotic manipulators that can handle rigid body contacts with friction. It is based on the time-stepping method presented by Anitescu and Potra, but the implementation required several modifications to allow for joint limits and error correction, as well as systems to control the joint motor forces and to perform collision detection and contact determination for complex polyhedral bodies. We have also demonstrated our simulator with a realistic grasp formation example that relies heavily on the accurate computation of contact forces. While we feel the system is quite useful for others in its current state, there are a few pieces we would like to improve.

The numerical integration scheme is currently only first order. After further optimization of the matrix computations, we plan to implement a higher order Runge-Kutta scheme which will give us additional accuracy and decrease the size of the error correction forces.

The current trajectory generator is quite simplistic and does not avoid singularities or joint force limits. We would like to implement a more sophisticated trajectory generation scheme as well as a path planner that could plan an approach that avoids obstacles given a desired grasp and a starting point.

One problem with using only PD control is that the actual position of a joint never matches the desired position exactly, so in the future we plan to add a feed-forward component that would compute the inverse dynamics of the system using the recursive Newton-Euler formulation. This should allow more accurate control.

References

[1] M. Anitescu and F. A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997.

[2] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, pages 292–352, October 1993.

[3] P. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, Mar. 1996.

[4] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[5] H. M. Hilber, T. J. R. Hughes, and R. L. Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering and Structural Dynamics*, 6:283–292, 1977.

[6] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Dept. of Computer Science, University of North Carolina, Chapel Hill, 1999.

[7] A. T. Miller and P. K. Allen. Graspit!: A versatile simulator for grasping analysis. In *Proc. of the ASME Dynamic Systems and Control Division*, volume 2, pages 1251–1258, Orlando, FL, 2000.

[8] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, Dept. of Computer Science, University of California at Berkeley, 1996.

[9] D. C. Ruspini and O. Khatib. Collision/contact models for the dynanmic simulation and haptic interaction. In *Proc. of the Ninth International Symposium of Robotics Research, ISRR'99*, pages 185–194, Snowbird, UT, 1999.

[10] A. Speck and H. Klaeren. RoboSiM: Java 3D robot visualization. In *IECON '99 Proceedings*, pages 821–826, 1999.

[11] M. R. Stein and S. Falchetti. A new graphics simulator for RCCL and its use in undergraduate robotics instruction. In *Proceeding 8th Intl. Conf. on Advanced Robotics, ICAR '97*, pages 825–830, 1997.

[12] D. Stewart and J. Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In *Proc. of the 2000 IEEE Intl. Conf. on Robotics and Automation*, pages 162–169, 2000.