# Extending Open Dynamics Engine for Robotics Simulation

Evan Drumwright[1], John Hsu[2], Nathan Koenig[2], and Dylan Shell[4]

[1] George Washington University, Washington DC, USA
drum@gwu.edu
[2] Willow Garage, Menlo Park, CA, USA
johnhsu@willowgarage.com, nkoenig@willowgarage.com
[3] Texas A&M University, College Station, TX, USA
dshell@cs.tamu.edu

**Abstract.** Open Dynamics Engine (ODE) is the most popular rigid-body dynamics implementation for robotics simulation applications. While using it to simulate common robotic scenarios like mobile robot locomotion and simple grasping, we have identified the following shortcomings each of which adversely affect robot simulation: lack of computational efficiency, poor support for practical joint-dampening, inadequate solver robustness, and friction approximation via linearization. In this paper we describe extensions to ODE that address each of these problems. Because some of these objectives lie in opposition to others—*e.g.,* speed versus verisimilitude—we have carried out experiments in order to identify the trade-offs involved in selecting from our extensions. Results from both elementary physics and robotic task-based scenarios show that speed improvements can be gained along with useful joint-dampening. If one is willing to accept an execution time cost, we are able to represent the full-friction cone, while simultaneously guaranteeing a solution from our numerical solver.

**Keywords:** Physical Simulation, Dynamics, Locomotion and Grasping.

## 1 Introduction

Simulation serves as an important tool for the development, programming, and testing of robots and their control software. We anticipate that in the coming years the significance of simulation will grow as it becomes an essential part of robotic design, debugging and evaluation processes. In retrospect, the inadequacy of current simulation technologies will likely be identified as one of early impediments to the field's progress and an initial failure to reach high standards of repeatability and scientific rigor. Indeed, current algorithms for physical simulation have several shortcomings which limit their use for problems of practical interest to the roboticist. In this paper we will describe and go some way toward addressing four such shortcomings in a popular rigid-body dynamics implementation for robotics simulation applications.

N. Ando et al. (Eds.): SIMPAR 2010, LNAI 6472, pp. 38–50, 2010.

A roboticist needs frustratingly little imagination in order to encounter serious flaws in current physical simulation. A scenario as simple as driving a mobile robot on a plane can be challenging if the user requires that frictional effects between tire and ground be realistically modeled. Several approximations are made in such a scenario, each of which affects the quality of the resultant output. Computational intractability, inefficient implementation, or both, mean that a user desiring real-time (or faster than real-time) response must sacrifice the level of detail modeled: but coarse polyhedral approximations and large integrator step-sizes influence the fidelity of the output. Perhaps worst of all, these influences operate in ways that remain poorly understood.

Although specialized algorithms, libraries and solvers exist to address these and other particular problems, in practice few have been incorporated in widely used robot simulators. Since most robot simulators employ external libraries and third-party code to handle simulation of the physical dynamics and contact mechanics, improvements usually occur when those code-bases are improved. Thus far, few observations of problems arising specifically in simulation of robots have resulted in extensions and modifications of these general purpose (and sometimes game-oriented) physical dynamics libraries. Modification of one such library is the approach we have employed in this paper.

Using the current version of the Open Dynamics Engine (ODE) [27] via Gazebo [18] to simulate mobile robot locomotion and simple grasping by a manipulator, we encountered and identified the following four distinct issues:

1. Currently, significant computational resources are required by ODE. Simulated execution is often significantly slower than real-time—and can be slower by orders of magnitude. The resultant long running times ultimately undermine the simulator's utility: lack of interaction makes for an inconvenient environment for iterative processes, running time limits the number of experiments that can be carried out, and physical hardware instantiations are required sooner in the design process than would be ideal.
2. The joint-dampening approach currently employed does not adequately model joints found in real robots and their physical mechanisms. Realistic simulation would require the parameter settings to be near the upper limits of the viscous friction joint dampening constant, which is not easily determined as a constant before run-time. This limits both the types of robots that can be simulated, and the sorts of environmental interactions (and hence tasks) that can be modeled.
3. The true friction cone is currently approximated by a linearized version which introduces orientation specific artifacts. These artifacts limit the set of physical phenomena that can be reproduced at a level adequate for robot applications. The default box-friction model results in highly non-isotropic frictional effects that can drastically alter the motions of mobile robots, grasped objects, vibrating items, *etc.*
4. Currently ODE employs linear complementarity problem (LCP) [10] solver to ensure forces satisfy non-interpenetration and joint-constraints. The solver does not always find a solution, which causes the constraints to be violated

within the simulation run. This produces in nonphysical behavior and a state from which further recovery is ill-defined.

We have implemented additions to ODE in order address these issues: extensions enabling multi-threaded execution of *quickstep*, iterative updates for joint dampening, and a convex-optimization solver. We also report on experiments that we have conducted in order to validate the correctness, assess the effectiveness of the methods, and determine the trade-offs involved.

## 2 Related Work

While no single robot simulator has yet emerged as a standard, many existing simulators make use of ODE as a library. By addressing shortcomings in this underlying piece of software, we believe one is most effective at improving the state of robotic simulation.

### 2.1 ODE for Simulating Robots

Open Dynamics Engine (ODE) is the most popular rigid-body dynamics implementation for robotics simulation. Two of the most widely used simulators are the open source Gazebo [18] and the commercial Webots [23], together they are used in hundreds of institutions and laboratories. Both make direct use of ODE for their dynamics simulation. Both are general robotics simulators: they model a range of robots. OpenSimulator [17] is similar in this regard, as is the book by Demur [12] devoted to the particularities of using ODE for simulation of a range of robots.

Several simulators intended specifically for simulating RoboCup soccer make use of ODE, including SPARK [26], SimRobot [19], UchilSim [33], and ÜberSim [7]. Virtual-RE [8] is an ODE-based simulator focused on soccer playing humanoid robots. Humanoid and bipedal robot simulators that employ ODE across a range of other tasks include the iCub Simulator [30], that of Lee and Oh [20], Sugiura and Takahashi [29], Moores and MacDonald [24], and Lima et al. [21]. ODE has also been used to model particular robots for synthesis applications *e.g.,* Wolff and Wahde [31] model physics via ODE to enable genetic programming to evolve controllers for biped locomotion. Finally, several groups have developed pluggable, or configurable simulations that cite ODE as a suitable module, most notably, MuRoSimF[15] and VISUM [14].

### 2.2 Other Methods for Robotic Simulation

Also worth mentioning, Bullet [11] has an implementation of the same PGS algorithm in ODE. Preliminary test runs for simple constrained dynamics through Gazebo showed similar constraint solving performances.

Comparisons for various simulators including popular physics engines such as Bullet [11] and PhysX[9] can be found in previous benchmarking efforts[5]. Unfortunately for robotics research and engineering, many physics engines have been developed and benchmarked with animation and gaming applications in mind[32], imperceptible error tolerances may not alway be the goal for many robotic simulations.

# 3 Approach

## 3.1 Current ODE Details

Smith [27] provides detail on the current version of ODE; this is an overview.

**Governing Equations.** In general, rigid body dynamic simulators attempt to solve the constrained Newton-Euler equation

$$\mathbf{M}\,\mathbf{a} = \mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{constraint}} + \mathbf{f}_{\text{damp}} \tag{1}$$

for systems of dynamic rigid bodies, where $\mathbf{f}_{\text{ext}}$ denotes the external applied forces, $\mathbf{f}_{\text{constraint}}$ denotes the constraint forces and $\mathbf{f}_{\text{damp}}$ denotes the viscous joint damping forces. ODE takes the approach of posing the governing equation (1) as an LCP problem in the maximal coordinate system, solving for constraint satisfying impulses for the rigid bodies connected by joints in simulation.

## 3.2 Speeding Up ODE: Quickstep Implementation

ODE Quickstep uses the projected Gauss-Seidel (PGS) method with Successive Overrelaxation (SOR) [3, 2].

Given that rigid body constraints are described by a constraint Jacobian $\mathbf{J}$ such that

$$\mathbf{J}\mathbf{v} = \mathbf{c} \tag{2}$$

The following mixed complementarity formulation [4] results:

$$\left[\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\right]\lambda = \frac{\mathbf{c}}{\Delta t} - \mathbf{J}\left[\frac{\mathbf{v}^n}{\Delta t} + \mathbf{M}^{-1}\left(\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{damp}}\right)\right], \tag{3}$$

where $\lambda \geq 0$ and $\mathbf{c} \geq 0$ for unilateral contacts.

During an update step of equation 3, the Jacobians $\mathbf{J}$ are treated explicitly (held constant).

Equation (3) can be written as a simple system of linear equations

$$\mathbf{A}\lambda = \mathbf{b} \tag{4}$$

where

$$\mathbf{A} = \left[\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\right], \qquad \mathbf{b} = \frac{\mathbf{c}}{\Delta t} - \mathbf{J}\left[\frac{\mathbf{v}^n}{\Delta t} + \mathbf{M}^{-1}\left(\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{damp}}\right)\right]. \tag{5}$$

Solve for the unknowns $\lambda$ in (3) using Gauss-Seidel written in delta form,

$$\delta_i = \frac{b_i}{A_{ii}} - \sum_{j=1}^{N_{constraints}} \frac{A_{ij}}{A_{ii}}\lambda_j, \text{ where } \lambda_i^{n+1} = \delta_i^n + \lambda_i^n \text{ for } i = 1, \cdots, N_{\text{constraints}}. \tag{6}$$

With SOR, a relaxation constant $\omega$ is introduced. The solution update becomes

$$\hat{\lambda}^{n+1} = (1-\omega)\,\lambda^n + \omega\lambda^{n+1}, \tag{7}$$

or

$$\hat{\lambda}^{n+1} = \lambda^n + \omega\left(\delta^n\right). \tag{8}$$

Thus, equation (6) is scaled by $\omega$

$$\delta_i = \omega\left(\frac{b_i}{A_{ii}} - \sum_{j=1}^{N_{\text{constraints}}} \frac{A_{ij}}{A_{ii}}\lambda_j\right). \tag{9}$$

**Additional implementation notes.** For efficiency, formulate the desired solution in the form of acceleration, denoted by

$$\mathbf{a}_c = \mathbf{M}^{-1}\mathbf{f}_c = \mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\lambda \tag{10}$$

then $\lambda$ updates in equation (9) becomes

$$\delta_i = \omega\left(\frac{b_i}{A_{ii}} - \frac{\sum_{k=1}^{N_{\text{DOF}}} J_{ik}a_{c_k}}{A_{ii}}\right) \text{ for } i = 1, ..., N_{\text{constraints}}. \tag{11}$$

At every iteration, for each $i$ update above, constraint accelerations $a_{c_k}$ are updated in the following manner:

$$a_{c_k}^{n+1} = a_{c_k}^n + G_{ki}\delta_i^n \text{ for } k = 1, ..., N_{\text{DOF}} \tag{12}$$

and where $\mathbf{G}$ is defined as

$$\mathbf{G} \equiv \mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}. \tag{13}$$

**Solution Projection.** Lastly, to enforce inequality constraints, each intermediate solution $\lambda_i^{n+1}$ is projected into its corresponding solution space depending on the type of constraint specified. For contact constraints, the constraint force corresponding to the normal direction of the contact,

$$f_{\text{constraint}\,i} = \mathbf{J}_i^{\mathsf{T}}\lambda_i, \tag{14}$$

is required to be push the bodies apart (no stiction), *i.e.,*

$$\lambda_i \geq 0, \tag{15}$$

where $\mathbf{J}_i^{\mathsf{T}}$ denotes the transpose of the $i$-th column of $\mathbf{J}$. And the tangential friction coefficients are truncated into its respective solution space depending on the friction coefficient,

$$f_{\text{fric}} \leq |\mu|\, f_{\text{normal}} \tag{16}$$

**Parallel Jacobi Gauss-Seidel Iterations.** Many past attempts at parallelizing iterative LCP solvers exist. Some examples are [22] [13] [16]. The approach taken in this paper simply performs update of individual row of the Gauss-Seidel iterations in parallel using latest available information. By updating multiple rows simultaneously, we have effectively turned subsets of the Gauss-Seidel algorithm into Jacobi iterations. Given that Jacobi's method has a tighter stability bound

than Gauss-Seidel iterations, runtime parameters must be tuned to prevent numerical instabilities. If the constraint matrix is well conditioned and diagonally dominant, the solution will converge. Otherwise, the user has to decrease the relaxation variable $\omega$ or increase ODE's *CFM* parameter to stabilize the system.

We observe that non-iterative approaches exist in the literature and that these can have speed advantages. Methods like those of Baraff [4] can solve acyclic arrangements in linear time. We note, however, that cyclic constraints occur frequently in robotics research, and thus we believe iterative methods are required in the general case.

## 4 Viscous Joint Damping

In the current *quickstep* implementation, ODE does not support simulation of viscous joint damping. The user is left with the option to apply joint damping explicitly outside of ODE, where the joint velocity from previous time step is used to compute the viscous joint damping force within the current time step,

$$\mathbf{f}_{\text{damp}}^{n+1} = k\mathbf{v}_{\text{joint}}^{n}. \tag{17}$$

Physical joints on real robots frequently have high mechanical reduction ratio leading to high viscous damping coefficients and low efficiency. Simulating large viscous damping forces leads to numerical stiffness for the physical system, requiring very small time step sizes and poor simulation performance. On the other hand, solving for damping forces implicitly is expensive. As a compromise, the forces can be updated within each PGS iteration with minimal loss in computational efficiency and higher tolerated viscous damping coefficient for a given fixed time step size.

### 4.1 Convex Optimization

ODE emphasizes speed and stability over physical accuracy Smith [27]. We describe here a method we term the convex optimization model, which models the complete friction cone rather than a linearized version of it.

The convex optimization model is formed and solved in two phases. Strictly speaking, this decomposition is unnecessary, though fast solution methods are admitted in this manner. The first phase consists of solving a *linear complementarity problem* (LCP). Much of the notation and problem setup will be similar to that presented in Anitescu and Potra [1]. Given the contact normal twist matrix $\mathbf{N}$, bilateral constraint Jacobian $\mathbf{J}$, lower and upper joint limit Jacobians $\mathbf{J}_l$ and $\mathbf{J}_h$, vector of external forces $\mathbf{k}$ and step size $h$, we solve the LCP below:

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}^{\mathsf{T}} & -\mathbf{N}^{\mathsf{T}} & -\mathbf{J}_l^{\mathsf{T}} & -\mathbf{J}_h^{\mathsf{T}} \\ \mathbf{J} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{N} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{J}_l & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{J}_h & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v}^{n+1} \\ \mathbf{c}_v \\ \mathbf{c}_n \\ \mathbf{c}_l \\ \mathbf{c}_h \end{bmatrix} + \begin{bmatrix} -\mathbf{M}\mathbf{v}^{n+1} - h\mathbf{k} \\ -\mathbf{j}_c \\ -\mathbf{n}_c \\ -\mathbf{j}_l \\ -\mathbf{j}_h \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{w} \\ \mathbf{x} \\ \mathbf{y} \end{bmatrix} \tag{18}$$

for the variables $\mathbf{v}^{n+1}$ (the new vector of generalized velocities after all impulses are applied), $\mathbf{c}_v$ (the bilateral constraint impulses), $\mathbf{c}_n$ (the magnitudes of impulses in the normal directions), $\mathbf{c}_l$ (the magnitudes of impulses applied at the lower joint limits), $\mathbf{c}_h$ (the magnitudes of impulses applied at the upper joint limits) and the (unused) slack variables $\mathbf{w}$, $\mathbf{y}$, and $\mathbf{z}$. Note that $\mathbf{c}_n$ and $\mathbf{w}$ are complementary, as are $\mathbf{c}_l$ and $\mathbf{x}$ and $\mathbf{c}_h$ and $\mathbf{y}$; in notation, $\mathbf{w}^\mathsf{T}\mathbf{c}_n = \mathbf{x}^\mathsf{T}\mathbf{c}_l = \mathbf{y}^\mathsf{T}\mathbf{c}_h = 0$.

The vectors $\mathbf{j}_c$, $\mathbf{n}_c$, $\mathbf{j}_l$, and $\mathbf{j}_h$ represent the amount of joint constraint violation, contact interpenetration, and joint limit violation already present in the simulation. In an ideal simulation environment, the vectors $\mathbf{j}_c$, $\mathbf{n}_c$, $\mathbf{j}_l$ and $\mathbf{j}_h$ would be zero vectors. In reality—due to ODE's fixed step sizes—joint constraints will be violated, contact interpenetration will occur, and joint limits will be violated. These vectors are provided automatically by ODE's joint constraint functions.

We now describe how to efficiently solve the LCP above. We can represent the matrix in (18) by blocks to yield $\begin{bmatrix} \mathbf{G} & -\mathbf{A} \\ \mathbf{A}^\mathsf{T} & \mathbf{0} \end{bmatrix}$, where $\mathbf{G} = \begin{bmatrix} \mathbf{M} & \mathbf{J}^\mathsf{T} \\ \mathbf{J} & \mathbf{0} \end{bmatrix}$ and $\mathbf{A} = \begin{bmatrix} -\mathbf{N}^\mathsf{T} & -\mathbf{J}_l^\mathsf{T} & -\mathbf{J}_h^\mathsf{T} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$.

We can then solve the LCP $(\mathbf{q}, \mathbf{S})$ (using the notation of Cottle et al. [10]), where $\mathbf{S} = \mathbf{A}^\mathsf{T}\mathbf{G}^{-1}\mathbf{A}$ and $\mathbf{q} = \begin{bmatrix} \mathbf{0} \\ -\mathbf{j}_l \\ -\mathbf{j}_h \end{bmatrix} + \mathbf{A}^\mathsf{T}\mathbf{G}^{-1}\begin{bmatrix} -\mathbf{M}\mathbf{v}^n - h\mathbf{k} \\ -\mathbf{j}_c \end{bmatrix}$, assuming that $\mathbf{G}$ is invertible[1] To compute the inverse of $\mathbf{G}$, we use the Schur-complement (see Nocedal and Wright [25]):

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}^\mathsf{T} \\ \mathbf{J} & \mathbf{0} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{C} & \mathbf{E} \\ \mathbf{E}^\mathsf{T} & \mathbf{F} \end{bmatrix} \text{ where } \begin{array}{l} \mathbf{F} = -\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}\,-1} \\ \mathbf{E} = -\mathbf{M}^{-1}\mathbf{J}^\mathsf{T}\mathbf{F} \\ \mathbf{C} = \mathbf{M}^{-1} - \mathbf{E}\mathbf{J}\mathbf{M}^{-1}. \end{array} \tag{19}$$

Solving the LCP $(\mathbf{q}, \mathbf{S})$ yields the solution variables $\mathbf{c}_n$, $\mathbf{c}_l$, and $\mathbf{c}_h$. Plugging these variables back into (18) yields the equality constrained variables. The advantage of solving this LCP, compared to the LCP used in ODE's original solver, is that the former problem is convex and much more easily solvable. We use a gradient projection method (*c.f.* Nocedal and Wright [25]) which is always able to provide a solution to the LCP.

Once the LCP has been solved, a frictionless, energy conserving solution to the system has been obtained. Friction can be added to the system using a nonlinear convex optimization model; the initial feasible point for the model is the solution to the MLCP determined above. The advantage of adding friction with such a model is that the convex optimization solver can be terminated early with only a loss of solution accuracy: energy will only be removed from the system.

[1] $\mathbf{G}$ is invertible as long as $\mathbf{J}$ is of full row rank, due to positive-definiteness of $\mathbf{M}$; we use a SVD-based pseudo-inverse which avoids this issue.

The nonlinear model consists of optimizing the following problem:

$$
\begin{aligned}
\text{Optimize:} \quad & \tfrac{1}{2}\mathbf{v}^{n+1}\mathbf{M}\mathbf{v}^{n+1} && \text{(Minimize kinetic energy)} \\
\text{Subject to:} \quad & \mathbf{N}\mathbf{v}^{n+1} \geq \mathbf{n}_c && \text{(Noninterpenetration constraint)} \\
& \mathbf{J}\mathbf{v}^{n+1} = \mathbf{j}_c && \text{(Bilateral constraint)} \\
& \mathbf{J}_l\mathbf{v}^{n+1} \geq \mathbf{j}_l && \text{(Lower joint limit)} \\
& \mathbf{J}_h\mathbf{v}^{n+1} \geq \mathbf{j}_h && \text{(Upper joint limit)} \\
& \mathbf{c}_n \geq \mathbf{0} && \text{(Compressive constraint)} \\
& \mathbf{1}^{\mathsf{T}}\mathbf{c}_n \leq \kappa && \text{(Compressive limit constraint)} \\
& \mu_i^2 c_{n_i}^2 \geq c_{s_i}^2 + c_{t_i}^2 && \text{(Friction cone constraint)}
\end{aligned}
$$

where $\mathbf{v}^{n+1} = \mathbf{M}^{-1}(\mathbf{N}^{\mathsf{T}}\mathbf{c}_n + \mathbf{J}^{\mathsf{T}}\mathbf{c}_j + \mathbf{J}_l^{\mathsf{T}}\mathbf{c}_l + \mathbf{J}_h^{\mathsf{T}}\mathbf{c}_h + \mathbf{S}^{\mathsf{T}}\mathbf{c}_s + \mathbf{T}^{\mathsf{T}}\mathbf{c}_t + h\mathbf{k}) + \mathbf{v}^n$.
The matrices $S$ and $T$ are the contact tangential twist matrices.

This nonlinear convex optimization problem exhibits the same $(O(N^3))$ complexity as the first phase of the model, though a significantly slower running time. Nevertheless, a full friction cone is obtained through the conic constraint above, and a $\epsilon$-suboptimal solution to the model can be obtained with superlinear convergence using a primal dual interior-point solver [6].

## 5 Experiments

A variety of different scenarios were considered, ranging from elementary physics based experiments to complex, application oriented robotics scenarios.

### 5.1 Ball Drop

An initially stationary 1kg sphere with unit radius is dropped from a height of 5m onto a plane. Figure 1 plots the vertical position of the ball. Both *quick*

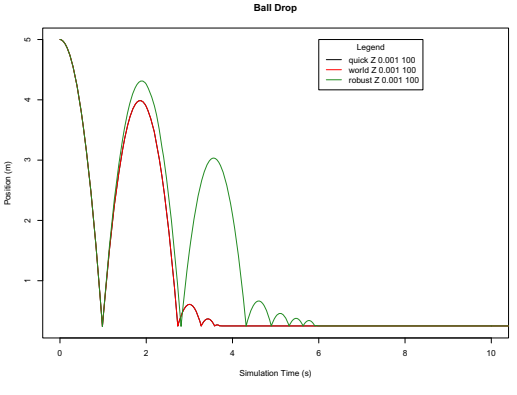
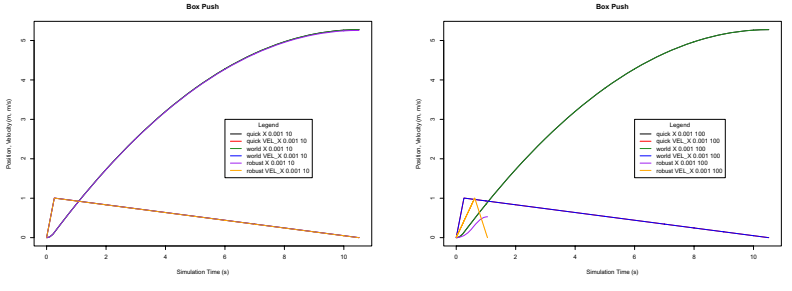



**Fig. 1.** Vertical (Z) position falling ball over time; comparison of methods, $step\_iterations = 100$

and *world* (which is the standard ODE function) show the same behavior. The *robust* method exhibits qualitatively similar behavior; the difference could be explained by different interpretations of the coefficient of restitution parameter, and the result suggests that a straightforward function may relate the parameter for the first two cases, to a different value which produces identical behavior in the *robust* method.

### 5.2 Box Push

A 0.1kg unit cube initially rests on a plane; a 0.2N force along **x** is applied until the box reaches a speed of $1\text{ms}^{-1}$. The force is withdrawn and the box slides to a stop. Figure 2 shows a particularly interesting result. The *robust* method brings the object to rest sooner than either of the other two methods. Analysis of the particular impulses generated by the *robust* method are consistent and correct under the principle of maximum dissipation [28]. Comparing the left and right figures, shows that increasing the convex optimization solver's number of iterations does indeed remove greater energy, as shown in the different slopes. Comparison of this example to an implementation outside of ODE highlighted something else: the normal impulses produced by ODE in order to compensate for penetration (due its treatment which may permit some interpenetration) may be larger than absolutely necessary. This increases the apex height of friction cone, permitting larger tangential impulses, which in turn, affects the time to bring the box to a halt.



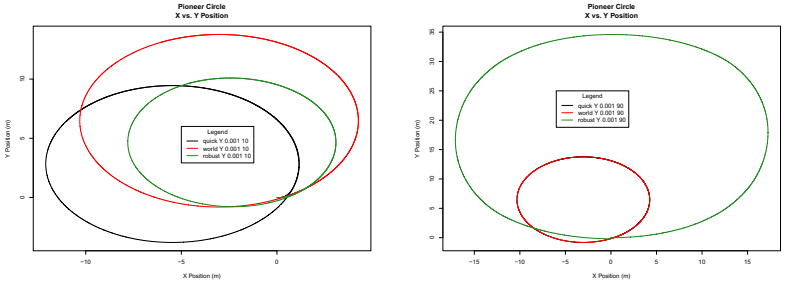

(a) Comparison of methods, $step\_iterations = 10$.

(b) Comparison of methods, $step\_iterations = 100$.

**Fig. 2.** Box Push Scenario

### 5.3 Pioneer Circle

A MobileRobots pioneer robot starts initially from rest. Torques of 2.2N·m and 2.1N·m are applied to the left wheel and right wheels respectively. The simulation terminates once 1.5 revolutions of the pioneer have been completed. Figure 3 illustrates the difference in circular performance. The difference in trajectory reflects friction cone approximation, and in the limit of large iterations, the robust method is expected to reproduce the true Coloumb friction behavior.



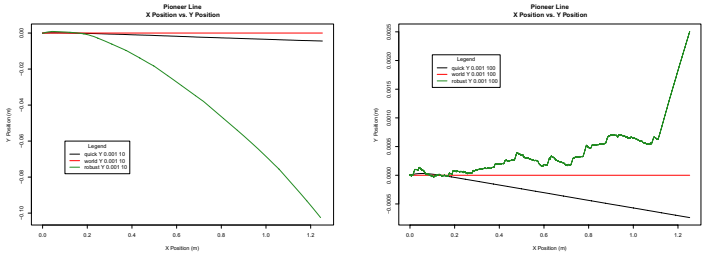

(a) Comparison of methods, $step\_iterations = 10$.

(b) Comparison of methods, $step\_iterations = 90$.

**Fig. 3.** Pioneer Circle Scenario

### 5.4 Pioneer Line

A MobileRobots pioneer robot starts initially from rest. Torques of 0.1N·m are applied to each wheel. The simulation terminates after 20s. Figure 4 shows the results from this scenario. Although unflattering, particularly to the *robust* method attention must be drawn to the comparatively small scale of the vertical axes in the plots, and hence the error.



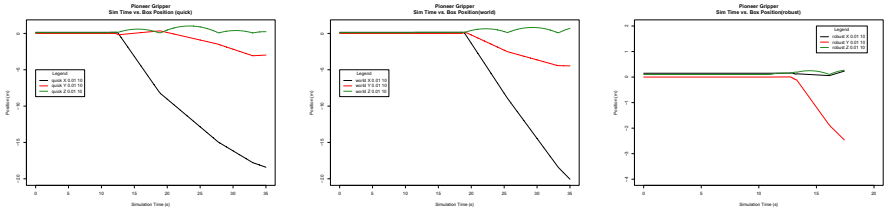

(a) Comparison of methods, $step\_iterations = 10$.

(b) Comparison of methods, $step\_iterations = 100$.

**Fig. 4.** Pioneer Line Scenario

### 5.5 Pioneer Gripper

A MobileRobots pioneer robot is simulated as it applies forces to a box (resting on the ground plane) with its paddle-style gripper. The box, placed between the paddles, is held in place through application of 500N of force. Figures 5(a), 5(b), and 5(c) show the results of experiments run with $\Delta t = 0.01$s and $step\_iterations = 10$. All three methods result in instability in which the grasped box ends up moving away from the correct location. We observe a surprising qualitative similarity in the results of all methods.



(a) *quick* method. (b) *world* method. (c) *robust* method.

**Fig. 5.** Position of grasped box over time

### 5.6 Parallelization: Box Stacks

We simulated four stacks of six boxes 1,2,4,8 processors, each parallelizing the box stacks row-wise. Each process drove residuals to $10 \times 10^{-3}$ at every time step. Figure 6 shows the result.

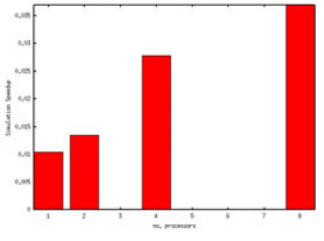


**Fig. 6.** Speedup of box stacks

## 6 Conclusion

We have focused on identifying problems that affect the Open Dynamics Engine (ODE), the rigid-body dynamics implementation used by the most popular robotics simulators, when attempting to simulate simple mobile robot locomotion and grasping. We have contributed the following: a parallelized implementation of ODE's quick step, viscous joint dampening, and a convex-optimization contact method. Our implementation has permitted us explore the behavior and identify some trade-offs and approximations involved (*e.g.,* robust method yields a true friction cone, but ODE may unphysically increase normal impulses) involved.

Much remains as future work: implementations of robust step method and parallel quick-step remain imperfect, and optimization remains possible. Nevertheless, we offer these tools as a mechanism for researchers needing their benefits.

## References


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

[2] Anitescu, M., Tasora, A.: An iterative approach for cone complementarity problems for nonsmooth dynamics. In: Computational Optimization and Applications (2008)

[3] Arechavaleta, G., López-Damian, E., Morales, J.L.: On the Use of Iterative LCP Solvers for Dry Frictional Contacts in Grasping. In: International Conference on Advanced Robotics, Munich, Germany, pp. 1–6 (June 2009)

[4] Baraff, D.: Linear-time dynamics using Lagrange multipliers. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH 1996, pp. 137–146 (1996)

[5] Boeing, A., Bräunl, T.: Evaluation of real-time physics simulation systems. In: Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia - GRAPHITE 2007, vol. 1(212), p. 281 (2007)

[6] Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)

[7] Browning, B., Tryzelaar, E.: ÜberSim: A Realistic Simulation Engine for Robot Soccer. In: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003), Melbourne, Australia (July 2003)

[8] Acosta Calderon, C.A., Mohan, R.E., Zhou, C.: Virtual-RE: A Humanoid Robotic Soccer Simulator. In: Proceedings of International Conference on Cyberworlds, Hangzhou, China, pp. 561–566 (September 2008)

[9] NVIDIA Corporation. PhysX SDK (2008), `http://www.nvidia.com/object/nvidia_physx.html`

[10] Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. Academic Press, Boston (1992)

[11] Coumans, E.: Bullet Physics Engine For Rigid Body Dynamics, `http://bulletphysics.org/`

[12] Demur, K.: Robot Simulation — Robot programming with Open Dynamics Engine. Morikita Publishing Co. Ltd., Tokyo (2007)

[13] Fijany, A.: New Factorization Techniques and Parallel O (Log N) Algorithms for Forward Dynamics Solution of Single Closed-Chain Robot Manipulators. Jet Propulsion Laboratory, California Institute of Technology

[14] Finkenzeller, D., Baas, M., Thüring, S., Yigit, S., Schmitt, A.: VISUM: A VR system for the interactive and dynamics Simulation of mechatronic systems. In: Fischer, X., Coutellier, D. (eds.) Research in Interactive Design: Proceedings of Virtual Concept 2003, Biarritz, France. Springer, Heidelberg (2003)

[15] Friedmann, M., Petersena, K., von Stryk, O.: Adequate motion simulation and collision detection for soccer playing humanoid robots. Robotics and Autonomous Systems 57(8), 786–795 (2009)

[16] Garstenauer, H., Kurka, D.I.D.G.: A unified framework for rigid body dynamics. Degree Paper (2006)

[17] Jung, D.: Opensim (2006), `http://opensimulator.sourceforge.net`

[18] Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Sendai, Japan, pp. 2149–2154 (September 2004)

[19] Laue, T., Spiess, K., Röfer, T.: SimRobot — A General Physical Robot Simulator and Its Application in RoboCup. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 173–183. Springer, Heidelberg (2006)
[20] Lee, J., Oh, J.H.: Biped Walking Pattern Generation Using Reinforcement Learning. International Journal of Humanoid Robotics 6(1), 1–21 (2009)
[21] Lima, J.L., Goncalves, J.C., Costa, P.G., Moreira, A.P.: Realistic Behaviour Simulation of a Humanoid Robot. In: Proceedings of 8th Conference on Autonomous Robot Systems and Competitions, Aveiro, Portugal (April 2008)
[22] Mangasarian, O.L., Leone, R.: Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems and linear programs. Annals of Operations Research 14(1), 41–59 (1988)
[23] Michel, O.: Webots: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems 1(1), 39–42 (2004)
[24] Moores, B.T., MacDonald, B.A.: A dynamics simulation architecture for robotic systems. In: Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA), Barcelona, Spain (April 2005)
[25] Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, Heidelberg (2006)
[26] Obst, O., Rollmann, M.: SPARK – A Generic Simulator for Physical Multiagent Simulations. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS (LNAI), vol. 3187, pp. 243–257. Springer, Heidelberg (2004)
[27] Smith, R.: ODE: Open Dynamics Engine
[28] Stewart, D.E.: Rigid-body dynamics with friction and impact. SIAM Review 42(1), 3–39 (2000)
[29] Sugiura, N., Takahashi, M.: Development of a Humanoid Robot Simulator and Walking Motion Analysis. In: Workshop Proceedings of SIMPAR, International Conference on Simulation, Modeling and Programming for Autonomous Robots, Venice, Italy, pp. 151–158 (November 2008)
[30] Tikhanoff, V., Fitzpatrick, P., Metta, G., Natale, L., Nori, F., Cangelosi, A.: An Open-Source Simulator for Cognitive Robotics Research: The Prototype of the iCub Humanoid Robot Simulator. In: Workshop on Performance Metrics for Intelligent Systems, National Institute of Standards and Technology, Washington DC, USA (August 2008)
[31] Wolff, K., Wahde, M.: Evolution of Biped Locomotion Using Linear Genetic Programming, ch. 16, pp. 335–356. Itech Education and Publishing, Vienna (October 2007)
[32] Yeh, T., Reinman, G., Patel, S.J., Faloutsos, P., Ageia Technologies: Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation. In: Simulation (2006)
[33] Zagal, J.C., Del Solar, J.R.: UchilSim: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, pp. 34–45. Springer, Heidelberg (2005)