Position-Based Time-Integrator for Frictional Articulated Body Dynamics

Zherong Pan[1] and Dinesh Manocha[2]
http://gamma.cs.unc.edu/PBAD/

Abstract— **We present a new time-integrator for modeling the frictional dynamics of articulated bodies. Our formulation represents the configuration of the articulated body using position variables and then uses those variables to model the friction forces between the articulated body and the environment. Our approach corresponds to a Newton-type optimization scheme that is guaranteed to converge so that it is stable with large timestep sizes. We evaluate the accuracy and stability of our time-integrator by comparing it with a conventional formulations based on the Newton-Euler equation and demonstrate the benefits on standard controller-optimization applications. We achieve $3 - 5$ times speedup over a Newton-Euler-based simulator on a CPU. Our approach can be easily parallelized on a GPU and results in additional $4 - 15$ times performance improvement.**

I. Introduction

Articulated body dynamics is widely used in various applications, including forward/inverse dynamics, kinodynamic motion planning, controller optimization, reinforcement learning, etc. These problems have been extensively studied in robotics and dynamics, and many solvers have been proposed based on the Newton-Euler equation. In fact, such solvers are also implemented in standard toolboxes such as Dart [20], Drake [32], Mujoco [33], etc. However, in certain applications including offline controller optimization, policy search [29], and sampling-based motion planning [23], [10], articulated body simulation is regarded as one of the major computational bottlenecks. This is because a large number of random samples are chosen simultaneously, where each sample corresponds to a trajectory generated by the simulator. Consequently, the total number of simulated timesteps can reach a few millions in a typical reinforcement learning application. One technique to reduce the number of timesteps is to use a larger timestep size. However, conventional articulated body simulators based on the Newton-Euler equation can become unstable under a large timestep size [2]. Another technique to improve the performance of articulated body simulators is to parallelize them on commodity parallel processors (e.g. GPUs).

Some of the widely used articulated body simulators perform time integration based on the Newton-Euler equation. During each simulation step, the equation is locally linearized and time-integrated using linear multistep algorithms such as the Runga-Kutta method. This algorithm can be adopted in the Euclidean space or the space of Lie algebra [6]. Unfortunately, it is not clear that the resulting solvers are stable under large timestep sizes. This is because that small timestep sizes ensure the sufficient accuracy of the local linearization [2]. It is also difficult to parallelize such solvers because they require sequential algorithms to resolve the frictional contacts that occur between the rigid body and obstacles in the environments. The handling of these contacts induces a set of complementarity conditions, and the exact resolution of these conditions is performed using sequential algorithms, such as a quadratic programming or a linear complimentary solver [31].

Our formulation is based on recent advances in position-based dynamics (PBD) [7], which represents an articulated body's configuration using only position variables in the Euclidean space. Based on this representation, PBD tends to be stable under arbitrarily large timesteps. This is because the PBD simulation step corresponds to a Newton-type optimization that is guaranteed to converge under certain conditions [14], [4], [27]. In addition, PBD solvers are relatively simple to implement because they only involve evaluation of objective functions. Due to these features, PBD has been successfully parallelized on a GPU [1]. However, prior PBD solvers have been mainly limited to rigid bodies or articulated models without frictional contacts.

Main Results: We present a novel PBD-based time integrator for frictional articulated body dynamics. We modify the optimization-based PBD formulation to time integrate friction forces in a fully implicit manner. We refer to our technique as position-based frictional dynamics (PBFD). PBFD is based on two novel techniques. We start from the complementarity conditions resulting from the conventional dry friction model [31] and approximate friction forces using position-based functions. We show that the position-based functions approximate dry friction forces arbitrarily well under certain conditions. However, these position-based functions are non-integrable and cannot be represented as objective energies. Therefore, they cannot be used directly in the original optimization-based PBD framework. To address this, our second technique includes the use of a new objective function that accounts for non-integrable forces, referred to as gradient-level objective function. Finally, we show that a conventional Newton-type optimizer will always bring the gradient level objective function to a local minimum, which corresponds to an articulated body configuration that satisfies Newton's laws at the next time instance.

We evaluate our PBFD formulation by comparing it with conventional PBD formulation and the Newton-Euler-based formulation on two standard simulation benchmarks: box sliding on ground and chain falling on slope. We compare their accuracy in terms of the friction force computation and stability under various timestep sizes. We also demonstrate that the accuracy of our simulator is sufficient for controller optimization applications using two reinforcement learning benchmarks. We show that the trained controller performs

[1]Zherong is with Department of Computer Science, the University of North Carolina. {zherong@cs.unc.edu} [2]Dinesh is with Department of Computer Science and Electrical & Computer Engineering, University of Maryland at College Park. {dm@cs.umd.edu}

well on both our PBFD simulator and the conventional Newton-Euler-based simulator. Our PBFD simulator results in $3-5$ times faster simulation on a CPU. The GPU parallelization of our gradient-level objective function evaluation results in an additional $4-15$ times speedup over the CPU implementation.

The rest of the paper is organized as follows. After reviewing related work in Section II, we formulate the conventional articulated body dynamics problem in Section III. Our PBFD formulation is described in Section IV-B and we present our efficient optimization algorithm in Section V. Finally, we highlight the performance on various benchmarks in Section VI.

II. Related Work

We briefly review related works in articulated body dynamics, frictional contact models, position-based dynamics, and applications to robotic control.

Articulated Body Dynamics: The governing equation of a rigid body is the Newton-Euler equation. An articulated body's governing equation is derived by formulating the Newton-Euler equation in the joint parameter space [12], [26]. Many discretization schemes have been proposed for this governing equation, and they vary in terms of accuracy and efficiency. The most widely used scheme in standard toolboxes is the simple linear multistep method [31], [34], which linearizes the governing equation in the Euclidean space. A more accurate set of methods is based on geometric integrators [18], [6], which linearize the governing equations in the space of Lie algebra. However, such geometric integrators have not been widely used in contact-rich scenarios.

Frictional Contact Models: Modeling frictional contacts is a central problem in dynamics. Prior methods to model friction can be categorized into complementarity-based models [31], [34], penalty-based models [8], and smooth approximate models [25], [28]. Complementarity-based models are the most popular in robotics due to their accuracy. However, solving for these complementarity conditions can be slow, and the underlying algorithms tend to be sequential. Fast approximate solvers for the complementarity conditions, such as iterative projection [5], staggered projection [17], and local methods [16], have been proposed, but their convergence properties are not well understood. Penalty-based models are much simpler to formulate and implement, but their accuracy is case-dependent and sensitive to the choice of parameters. The smooth approximate models [25], [28] formulate friction forces as a smooth and differentiable function and the resulting algorithms work well with continuous optimization techniques. However, we show that all the prior smooth approximate models may not be accurate because the friction forces can be underestimated.

Position-based Dynamics: PBD is regarded as a discretization scheme that represents all the variables and their finite differences using the articulated body's position in the Euclidean space [3]. PBD was originally proposed to model fluid dynamics [24] and elastic deformations [4]. In [28], PBD was extended to articulated body dynamics under minimal coordinates. A nice feature of PBD is that the underlying simulation algorithm corresponds to a numerical optimization [14], [4], [15] under certain conditions. Therefore, an off-the-shelf numerical optimizer can be used as a simulator, leading to simpler implementation and improved stability under large timestep sizes. However, it is still difficult for prior PBD algorithms to handle friction forces in a robust manner.

Robotic Control Applications: The recent surge in deep learning and statistical models has resulted in the development of new control algorithms for complex dynamic systems such as articulated humanoids [9], [19], [30]. A common point in these algorithms is that they train robust controllers by injecting noise into the system and optimize the controller performance in these noisy scenarios. Such noises can be injected in different ways, such as perturbations to the control signals, system dynamics, or observations. The magnitude of these noises is generally much larger than the discrepancy between an accurate simulator and a less accurate one. As a result, it is possible to use slightly less accurate, but more efficient PBD simulators for such applications.

III. Problem Formulation

In this section, we formulate the articulated body simulation problem. More details are given in [28], [31].

A. Newton-Euler's Equation

As illustrated in Figure 1 (a), an articulated body $\mathcal{A}$'s configuration is represented using joint parameters, θ. For each rigid body $\mathcal{R}$ in $\mathcal{A}$, we denote $V=\begin{pmatrix}v^T & w^T\end{pmatrix}^T$ as its linear/angular velocity and assume $V=J\dot{\theta}$. We denote $M^{6\times6}$ as the generalized mass matrix in the world coordinates, and $F=\begin{pmatrix}f^T & \tau^T\end{pmatrix}^T$ as the external force/torque. With these symbols, $\mathcal{R}$'s dynamics are governed by the Newton-Euler equation parameterized using θ and can be expressed as:

$$J^TMJ\ddot{\theta}+(J^TM\dot{J}+J^T\begin{pmatrix}0 & \\ & [\omega]\end{pmatrix}MJ)\dot{\theta}+J^TF=0, \quad (1)$$

where $[\omega]$ is the cross product matrix. Throughout the paper, we present our PBFD method using a single rigid body $\mathcal{R}$. The governing equations for the entire $\mathcal{A}$ can be derived by concatenation. For numerical simulation, the prominent method is to discretize Equation 1 into a sequence of time instances, $n\Delta t$, with uniform interval Δt. We denote the timestep indices using subscripts. For example, with first-order finite difference, we assume:

$$(J\dot{\theta})_{n+1}\approx J_n(\theta_{n+1}-\theta_n)/\Delta t. \quad (2)$$

B. Position-based Dynamics

If we evaluate a continuous point p on R, its position is the function $p(\theta)$ and Equation 2 is p's instantaneous velocity. Instead, PBD uses p's average velocity over Δt:

$$(J\dot{\theta})_{n+1}\approx(p(\theta_{n+1})-p(\theta_n))/\Delta t, \quad (3)$$

which is illustrated in Figure 1 (b). Note that Equation 2 is a linearization of Equation 3 at time $n\Delta t$, which is accurate only when $\theta_{n+1}-\theta_n$ is small. This linearization limits the ability of Equation 2 to handle large timestep sizes, while

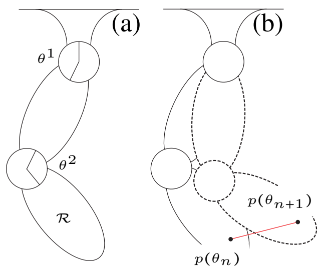


Fig. 1: (a): The configuration of $\mathcal{A}$ is represented using joint parameters θ at time instance $n\Delta t$. (b): At the time instance $(n+1)\Delta t$, $\mathcal{A}$ moves to a new configuration (dashed line). The average velocity of any point $p \in \mathcal{R}$ is defined as $(p(\theta_{n+1}) - p(\theta_n))/\Delta t$ (red line) in the PBD formulation.

Equation 3 can always be interpreted as the average velocity of p over the period of Δt in the Euclidean space, regardless of how large Δt is. Under this interpretation, the governing equation of $\mathcal{R}$ is derived by taking an integral over $\mathcal{R}$ as follows:

$$0 = \int_{\mathcal{R}} \frac{\partial p(\theta_{n+1})}{\partial \theta_{n+1}}^T \left[\rho \frac{(p(\theta_{n+1})-2p(\theta_n)+p(\theta_{n-1}))}{\Delta t^2} - f_p(\theta_{n+1})\right] dp, \quad (4)$$

where ρ is the mass density and $f_p(\theta)$ is the external force density, see [] for more details. The terms in the bracket are derived from Newton's second law for each point p and the multiplication by $\frac{\partial p(\theta)}{\partial \theta_{n+1}}^T$ on the left ensures that all the points move as one rigid body. A remarkable feature of Equation 4 is that its right-hand side is the derivative of the following objective function, $E(\theta)$:

$$\begin{aligned} E(\theta_{n+1}) &= I(\theta_{n+1}) + P(\theta_{n+1}), \\ I(\theta_{n+1}) &= \int_{\mathcal{R}} \frac{\rho}{2\Delta t^2} \|p(\theta_{n+1}) - 2p(\theta_n) + p(\theta_{n-1})\|^2 dp, \\ \frac{\partial P(\theta_{n+1})}{\partial \theta_{n+1}} &= -\int_{\mathcal{R}} \frac{\partial p(\theta_{n+1})}{\partial \theta_{n+1}}^T f_p(\theta_{n+1}) dp. \end{aligned} \quad (5)$$

In $E(\theta)$, the first term $I(\theta)$ models the inertia of $\mathcal{R}$, and the integral in $I(\theta)$ can be evaluated analytically (we refer readers to [28] for its derivation). The second term models external forces, and PBD assumes that $f_p(\theta)$ is the force of a conservative energy, $P(\theta)$.

In conclusion, simulation in the PBD framework is equivalent to the following optimization:

$$\theta_{n+1} = \underset{\theta}{\mathbf{argmin}}\, E(\theta), \quad (6)$$

which is easy to implement using an off-the-shelf numerical optimizer. In addition, the following lemma ensures that a local minimum can always be computed under some assumptions:

Lemma 3.1: If $P(\theta) \in \mathcal{C}^1$ is bounded from below, then a Newton-type solver will always bring $E(\theta)$ to a local minimum.

Proof: Since $I(\theta)$ is a composition of smooth functions (joint transformations), we have $I(\theta) \in \mathcal{C}^\infty$. We also have $I(\theta) \geq 0$. Therefore, combined with $P(\theta)$, $E(\theta) \in \mathcal{C}^1$ is also bounded from below. The convergence of a Newton-type solver under these conditions is shown, e.g., in Theorem 3.2 of [35]. ■

Lemma 3.1 combined with PBD's independence of local linearization provides a strong support for its stability under large timestep sizes.

C. Contact Handling

Since a robot frequently interacts with the environment, a robust articulated body simulation algorithm needs to handle frictional contacts. In this paper, we restrict ourselves to the dry friction model. Using Equation 1 and Equation 2, we simultaneously solve for the external force f and the velocity $J\dot{\theta}$ for each contact point. In this case, the dry friction model

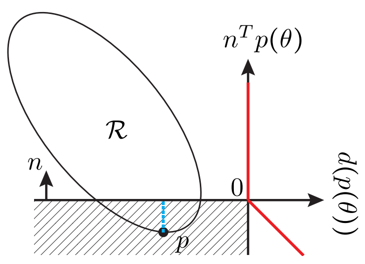


Fig. 2: The penetration depth, $d(p(\theta))$, at point p is the magnitude of the blue vector. We plot the change of $d(p(\theta))$ as the red curve, which is a piecewise smooth function. Here the obstacle is the hatched area and n is the unit outward contact normal.

can be conveniently formulated as two complementarity conditions [31]:

$$\begin{aligned} 0 \leq J^\perp \dot{\theta} \quad &\wedge \quad f^\perp \geq 0 \\ 0 \leq \mu f^\perp - \|f^\parallel\| \quad &\wedge \quad \|J^\parallel \dot{\theta}\| \geq 0, \end{aligned} \quad (7)$$

where $\perp$ and $\parallel$ are the normal and tangential components of the force or velocity, respectively. μ is the friction coefficient. The first condition ensures that normal forces are unilateral and are just sufficient for an inelastic collision. The second condition ensures that friction forces are in the frictional cone, and that they are on the cone boundary only when there is relative motion. However, modeling these complementarity conditions is difficult in the PBD framework because all external forces are modeled implicitly as conservative energies and are not solved explicitly. In addition, it is known that friction forces are not conservative forces.

IV. Position-based Frictional Dynamics (PBFD)

Since PBD uses position variables, our idea is to model contact forces at point p using only the position variables. We first discuss the modeling of normal forces, $f_p^\perp(\theta)$, in Section IV-A. Then we analyze previous inaccurate tangential force models used by PBD in Section IV-B. Finally, the core techniques of our PBFD formulation are derived in Section IV-C.

A. Normal Force Model for PBD

To represent $f_p^\perp(\theta)$ using position-based variables, we assume that $f_p^\perp(\theta)$ is a function of the penetration depth $d(p)$. In this paper, we use the following relationship:

$$f_p^\perp(\theta) \quad = \quad d(p(\theta))^\alpha n, \quad (8)$$

where n is the unit outward contact normal. α is a constant coefficient that dictates the speed of normal force growth as the penetration depth increases. In addition, we can modify the smoothness property of $f_p^\perp(\theta)$ by changing α. Equation 8 is fully compatible with the PBD framework because it has a conservative energy form:

$$P^\perp(\theta) = \frac{1}{\alpha+1} \int_{\mathcal{R}} d(p(\theta))^{\alpha+1} dp. \quad (9)$$

In practice, the integral in Equation 9 is replaced with a summation over the set of points found by a contact detector. It is easy to verify that Equation 8 and Equation 9 satisfy Equation 5. The remaining issue is that $d(p)$ is only piecewise smooth, and it is $\mathcal{C}^0$ globally as illustrated in Figure 2. Fortunately, the property used by Lemma 3.1 would still hold based on the following lemma:

Lemma 4.1: If $\alpha > 0$, then $d(p(\theta))^{\alpha+1} \in \mathcal{C}^1$, $P^\perp(\theta) \in \mathcal{C}^1$, and Lemma 3.1 holds when $P(\theta) = P^\perp(\theta)$ in Equation 5.

Proof: Since $d(p(\theta))^\alpha$ is piecewise smooth, its gradient is:

$$\frac{\partial d(p(\theta))^{\alpha+1}}{\partial\theta}=\begin{cases}-(\alpha+1)\left[\frac{\partial p(\theta)}{\partial\theta}^T n\right]d(p(\theta))^\alpha & d(p(\theta))>0\\ 0 & \text{otherwise}\end{cases}.$$

However, when $d(p(\theta))\to 0$ in the first case, $d(p(\theta))^\alpha\to 0$. Therefore, the point where $d(p(\theta))=0$ is also continuous and $d(p(\theta))^{\alpha+1}\in\mathcal{C}^1$. It is obvious that $P^\perp(\theta)\in\mathcal{C}^1$. Finally, Lemma 3.1 holds because $P^\perp(\theta)\geq 0$, i.e. this function is bounded from below. ■

However, it is more difficult to model tangential forces because they depend on both the tangential velocity and the normal force.

B. Tangential Force Model for PBD

We notice that the tangential force always acts to damp the velocity. Therefore, we could introduce a velocity damping force to approximately model the friction using the following conservative energy formulation:

$$\int_{\mathcal{R}}\|(I-nn^T)(p(\theta)-p(\theta_n))\|^{\beta+1}dp, \tag{10}$$

where $I-nn^T$ represents a projection matrix to the tangential plane and β is a constant coefficient with similar function as α. Note that every conservative force has an equivalent energy form, but an energy form does not necessarily correspond to a conservative force. For example, Equation 10 corresponds to a dissipative force pointing in the opposite direction of the tangential velocity. This is because Equation 10 acts to minimize the difference between $p(\theta)$ and $p(\theta_n)$, which is proportional to the velocity at p according to Equation 3. However, Equation 10 does not approximate the dry friction model because it applies the friction force even when p is not in contact, i.e. $f_p^\perp(\theta)=0$. To respect the fact that the magnitude of $f^\parallel$ is upper bounded by $\mu f^\perp$, a simple idea is to weight Equation 10 using $f^\perp$. Since $f^\perp$ is proportional to $d(p(\theta))$ in our formulation, we multiply the integrands in Equation 9 and Equation 10 to get the following conservative energy for tangential forces:

$$P^\parallel(\theta)=\int_{\mathcal{R}}\|(I-nn^T)(p(\theta)-p(\theta_n))\|^{\beta+1}d(p(\theta))^{\gamma+1}dp, \tag{11}$$

where γ a constant coefficient with similar function as α. At this point, we can model both the normal and friction forces by setting $P(\theta)=C^\perp P^\perp(\theta)+C^\parallel P^\parallel(\theta)$ and Lemma 3.1 holds for this $P(\theta)$ if $\alpha,\beta,\gamma>0$, following a similar argument as that in Lemma 4.1. Here we introduce two additional constant coefficients $C^\perp, C^\parallel$. Variants of this $P(\theta)$ have been used in [28] for articulated body simulation.

However, one of our key observations is that Equation 11 is still not accurate because it does not approximate the dry friction model in any sense, and the friction forces are considerably underestimated. To see this, we evaluate the partial derivatives, $\frac{\partial P^\parallel(\theta)}{\partial\theta}$, to derive the corresponding forces of $P^\parallel(\theta)$:

$$\begin{aligned}\frac{\partial P^\parallel(\theta)}{\partial\theta} &= -\int_{\mathcal{R}}\frac{\partial p(\theta)}{\partial\theta}^T(f_p^\parallel(\theta)+f_p^*(\theta))dp,\\ f_p^\parallel(\theta) &\triangleq -(\beta+1)(I-nn^T)(p(\theta)-p(\theta_n))\\ &\quad\|(I-nn^T)(p(\theta)-p(\theta_n))\|^\beta d(p(\theta))^{\gamma+1},\\ f_p^*(\theta) &\triangleq (\gamma+1)n\|(I-nn^T)(p(\theta)-p(\theta_n))\|^{\beta+1}d(p(\theta))^\gamma.\end{aligned} \tag{12}$$

From these equations, we find that $P^\parallel(\theta)$ corresponds to two force terms. The first force, $f_p^\parallel(\theta)$, is the desired tangential friction force. However, the second force, $f_p^*(\theta)$, is an undesired term. Intuitively, $f_p^*(\theta)$ means that a contact point will produce larger normal forces to $\mathcal{R}$ with larger tangential speed, which does not correspond to any physical phenomena. In our benchmarks, $f_p^*(\theta)$ produces large non-physical normal forces for fast moving $\mathcal{R}$. As a result, $f_p^*(\theta)$ will erroneously reduce $d(p(\theta))$, which in turn reduces $f_p^\parallel(\theta)$.

C. Corrected Tangential Force Model for PBFD

In this section, we present a corrected tangential force model. Our idea is to simply remove the second force, $f_p^*(\theta)$ which was a result from $P^\parallel(\theta)$, and retains only $f_p^\parallel(\theta)$. However, this treatment is incompatible with the prior PBD framework because $f_p^\parallel(\theta)$ alone is not integrable, and it cannot be written in an energy form. This difficulty leads us to develop a generalized version of PBD, called gradient-level PBD. Our core idea is that, instead of minimizing $E(\theta)$, we minimize $GE(\theta)\triangleq\frac{1}{2}\|\frac{\partial E(\theta)}{\partial\theta}\|^2$. Indeed, under mild assumptions, these two objective functions are equivalent due to the following lemma:

***Lemma** 4.2:* If $P(\theta)\in\mathcal{C}^2$, then a Newton-type solver minimizing $GE(\theta)$ will converge to a local minimum of $GE(\theta)$. If $E(\theta)$ is locally convex, then a Newton-type solver minimizing $GE(\theta)$ will converge to zero, i.e. a local minimum of $E(\theta)$.

Proof: For the first claim, if $P(\theta)\in\mathcal{C}^2$, then $E(\theta)\in\mathcal{C}^2$, $\frac{\partial E(\theta)}{\partial\theta}\in\mathcal{C}^1$, and $GE(\theta)\in\mathcal{C}^1$. Combined with the fact that $GE(\theta)\geq 0$, the convergence of the Newton-type solver follows from Lemma 3.1. For the second claim, we have $\frac{\partial GE(\theta)}{\partial\theta}=\frac{\partial^2E(\theta)}{\partial\theta^2}\frac{\partial E(\theta)}{\partial\theta}$. On convergence, $\frac{\partial GE(\theta)}{\partial\theta}=0$, but $E(\theta)$ being locally convex implies that $\frac{\partial^2E(\theta)}{\partial\theta^2}$ is of full rank, so that we have $\frac{\partial E(\theta)}{\partial\theta}=0$ and $GE(\theta)=0$. ■

Lemma 4.2 imposes a stronger assumption on the smoothness of $P(\theta)$. However, Lemma 4.2 generalizes PBD to the case of non-integrable force terms while still casting the simulation as an optimization with convergence guarantee. This property satisfies our need to handle the non-integrable tangential force $f_p^\parallel(\theta)$. In summary, we propose to set $P(\theta)=C^\perp P^\perp(\theta)$, which only accounts for normal force. During each simulation step, we solve the following minimization problem:

$$\begin{aligned}GE(\theta) &= \frac{1}{2}\|\frac{\partial E(\theta)}{\partial\theta}-C^\parallel\int_{\mathcal{R}}\frac{\partial p(\theta)}{\partial\theta}^T f_p^\parallel(\theta)dp\|^2\\ \theta_{n+1} &= \underset{\theta}{\mathbf{argmin}}\,GE(\theta),\end{aligned} \tag{13}$$

which adds tangential forces, $f_p^\parallel(\theta)$, as a non-integrable term. We call this formulation the position-based frictional dynamics (PBFD) model. In order for Lemma 3.1 to hold for Equation 13, we need $P^\perp(\theta)\in\mathcal{C}^2$ and $f_p^\parallel(\theta)\in\mathcal{C}^1$. Following a similar argument as Lemma 4.1, this requires $\alpha>1,\beta,\gamma>0$.

D. PBFD and Dry Friction Model

Compared with the dry friction model, which has only one parameter μ, PBFD has five parameters $\alpha>1,\beta,\gamma>$

$0, C^{\perp}, C^{\parallel}$. These two models coincide when $\mu, C^{\perp}, C^{\parallel} \to \infty$ at the same time. In this case only static contact occurs, and both models ensure that the tangential velocity is exactly zero, because otherwise $GE(\theta) \to \infty$ in the PBFD model.

However, these two models would exhibit different behaviors in the case of sliding contacts. In this case, the dry friction model will apply a force $\|f^{\parallel}\| = \mu f^{\perp}$ independent of the relative tangential velocity between $\mathcal{R}$ and the obstacle. However, PBFD will apply a velocity dependent force $f_p^{\parallel}(\theta)$ due to the presence of $(p(\theta) - p(\theta_n))$ in Equation 12.

V. Optimization Algorithm for PBFD

We presented our PBFD formulation (Equation 13) in the previous section. In this section, we present our optimization algorithm based on that formulation and compare its performance with the original PBD algorithm (Equation 6). We consider three broad categories of optimization algorithms:

- LM optimizer [21] using gradient and accurate Hessian.
- LM optimizer [21] using gradient and JTJ-approximate Hessian (JTJ-LM).
- LBFGS optimizer [22] using only gradient.

In short, given an objective function $GE(\theta)$, an LM optimizer requires the user to provide both $\frac{\partial GE(\theta)}{\partial\theta}$, $\frac{\partial^2 GE(\theta)}{\partial\theta^2}$ and updates θ according to:

$$\theta \triangleq \theta - (\frac{\partial^2 GE(\theta)}{\partial\theta^2} + \lambda I)^{-1} \frac{\partial GE(\theta)}{\partial\theta}, \tag{14}$$

where λ is the diagonal regularization tuned by LM internally to ensure convergence. JTJ-LM takes the same steps but requires only an approximate Hessian denoted as $\overline{\frac{\partial^2 GE(\theta)}{\partial\theta^2}}$. There are many ways to approximate the true Hessian. In particular, JTJ-LM assumes that $GE(\theta)$ can be written as a sum of squared functions, and the Hessian is approximated by linearizing each function. For example, in the case of Equation 13, JTJ-LM uses the following approximate Hessian:

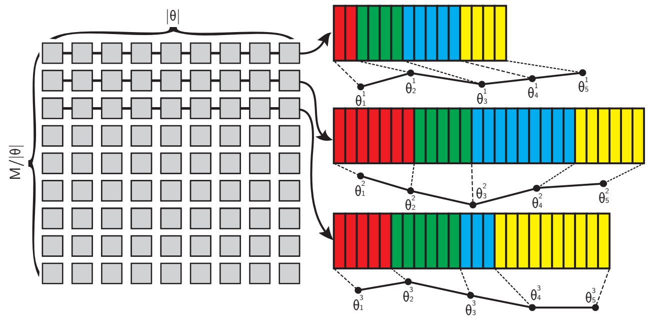
$$\overline{\frac{\partial^2 GE(\theta)}{\partial\theta^2}} = \frac{\partial}{\partial\theta}\left[\frac{\partial E(\theta)}{\partial\theta} - C^{\parallel}\int_{\mathcal{R}} \frac{\partial p(\theta)}{\partial\theta}^T f_p^{\parallel}(\theta)dp\right]^T \frac{\partial}{\partial\theta}\left[\frac{\partial E(\theta)}{\partial\theta} - C^{\parallel}\int_{\mathcal{R}} \frac{\partial p(\theta)}{\partial\theta}^T f_p^{\parallel}(\theta)dp\right],$$

where the term related to $\frac{\partial^3 E(\theta)}{\partial\theta^3}$ is ignored. As long as $\overline{\frac{\partial^2 GE(\theta)}{\partial\theta^2}}$ is symmetric and positive semi-definite, JTJ-LM will converge to a local minima. Finally, the LBFGS optimizer only requires the computation of $\frac{\partial GE(\theta)}{\partial\theta}$ and it approximates $\frac{\partial^2 GE(\theta)}{\partial\theta^2}$ internally.

A. GPU Parallelization for the JTJ-LM Algorithm

To further accelerate the performance of the JTJ-LM algorithm, we use GPU parallelization. Our parallel algorithm is based on the assumption that multiple trajectories must be computed simultaneously and independently so that a multi-threaded acceleration can be used to provide additional speedup. This assumption is true for reinforcement learning [9] and online feedback control [11] applications. Specifically, we assume that an application needs to compute a set of K trajectories, each from a different initial configuration θ_1^k, where $1 \le k \le K$. Our parallel algorithm runs on a GPU chip with M cores. Instead of having each core simulating a separate trajectory, we instead let $|\theta|$ cores work together to simulate a single trajectory, where $|\theta|$ is the number of DOFs in $\mathcal{A}$. In this way, the cost of matrix-matrix production and matrix inversion can be performed in parallel, and their cost can be reduced from $\mathcal{O}(|\theta|^3)$ to $\mathcal{O}(|\theta|^2)$, according to [13]. This formulation assumes $|\theta| \ll M$, which is true for most modern GPU models with more than 1000 cores. As a result, a single GPU chip can compute $\lfloor M/|\theta| \rfloor$ trajectories in parallel.

An additional improvement in our parallel algorithm is that, instead of having each group of $|\theta|$ cores evaluate Equation 14 until convergence, we have each group evaluate Equation 14 once during each call to the GPU. As a result, all the GPU core groups are performing the same computation and no core suffers from starvation while waiting for other cores. The entire picture of GPU scheduling is illustrated in Figure 3.



|θ|
M/|θ|
θ_1^1 θ_2^1 θ_3^1 θ_4^1 θ_5^1
θ_1^2 θ_2^2 θ_3^2 θ_4^2 θ_5^2
θ_1^3 θ_2^3 θ_3^3 θ_4^3 θ_5^3

Fig. 3: We illustrate our GPU scheduling algorithm. In this case, we compute three trajectories from three initial configurations, $\theta_1^{1,2,3}$. Each GPU core is a gray box, and every $|\theta|$ cores work together to simulate a single trajectory (shown as a black arrow). Each timestep in each trajectory involves evaluating Equation 14 multiple times until convergence. Each evaluation is illustrated as a colored block on the right. The evaluations used in a single timestep are marked with the same color. For example, it takes 2 iterations for the first trajectory to find θ_2^1, but it takes 7 iterations to find θ_2^2, etc. During each call to the GPU, we only compute one block for all the trajectories, instead of computing all the blocks with the same color.

B. Time Complexity

We first analyze the time complexity of each iteration in an optimization algorithm. The common steps of one iteration are to first compute the gradient and/or (approximate) Hessian and then solve the linear system. We know from [28] that, on both CPU and GPU, computing $\frac{\partial E(\theta)}{\partial\theta}$ has complexity $\mathcal{O}(|\theta|)$ and computing $\frac{\partial^2 E(\theta)}{\partial\theta^2}$ has complexity $\mathcal{O}(|\theta|^2)$. Computing $\frac{\partial^3 E(\theta)}{\partial\theta^3}$ has complexity $\mathcal{O}(|\theta|^3)$ at least. Solving the linear system takes $\mathcal{O}(|\theta|^3)$ on CPU and $\mathcal{O}(|\theta|^2)$ on GPU. Note that a matrix inversion using branch-induced sparsity has complexity $\mathcal{O}(|\theta|^2)$ on CPU [12], but it cannot be used to solve a general JTJ form matrix. Finally, computing derivatives of $GE(\theta)$ will require higher order derivatives of $E(\theta)$. For example, the computation of $\frac{\partial GE(\theta)}{\partial\theta}$ involves evaluating $\frac{\partial^2 E(\theta)}{\partial\theta^2}$. The time complexities of all six cases are summarized in Table I. Of the six choices, the costliest one is to use LM for solving Equation 13, which involves complex third order derivative evaluation and is clearly

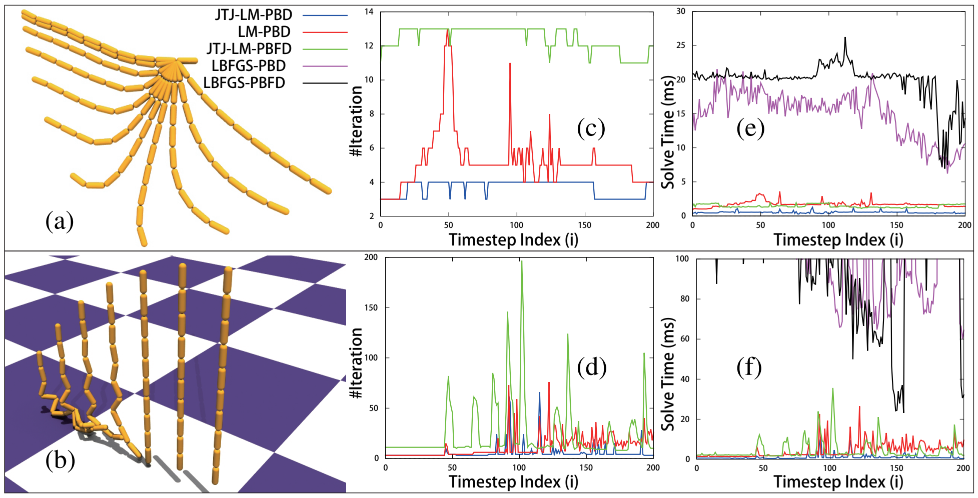


Fig. 4: The two benchmarks we use to evaluate the performance of different optimizers. (a): fixed 10-link chain with no contacts. (b): floating 10-link chain falling on the ground. We evaluate the five different optimization algorithms listed in Table I, except LM+Equation 13. For each of the five choices, we plot the number of iterations taken by the optimizer to convergence (c,d), and the computational time to finish the optimization (e,f) for each timestep.

much costlier than other choices. Therefore, we exclude it from our comparisons. In summary, our PBFD formulation does require higher-order derivative computations than the PBD formulation, but the time complexity of the JTJ-LM algorithm is the same for both formulations.

As illustrated in Figure 4, we use two benchmarks to evaluate the performance of different optimizers. Our first benchmark is a 10-link chain under gravity with one end fixed, where no contact forces are involved, so that Equation 6 coincides with Equation 13. In our second benchmark, the 10-link chain has a floating base and falls on the ground under gravity, where contact forces are involved. The time cost of the five choices listed in Table I are plotted against timestep indices. From the plots in Figure 4, we can see that, although each iteration of LBFGS is faster in theory, many more iterations are needed (off the chart in Figure 4 (c,d)) and the overall performance of LBFGS is worse than that of LM (Figure 4 (e,f)). For PBD, JTJ-LM generally converges faster than LM because each iteration is cheaper. Finally, each iteration of PBFD is more costly than PBD using JTJ-LM but the difference is marginal. In Section VI, we always use JTJ-LM as our underlying optimizer.

VI. Results and Analysis

We use two implementations of our algorithm based on the multi-trajectory assumption. Our first implementation is on a 4-core CPU (Intel i7-4790 3.6G) of a desktop machine, where each core runs a single thread simulating a single trajectory. Note that each iteration of (JTJ-)LM optimization on the CPU takes $\mathcal{O}(|\theta|^3)$ due to a serial matrix inversion. Our second implementation is on a 3584-core GPU (Nvidia Titan-X) with every $|\theta|$ cores simulating a single trajectory. In this case, each iteration of (JTJ-)LM optimization takes $\mathcal{O}(|\theta|^2)$. In this section, we evaluate the accuracy and efficiency of the PBFD formulation using a set of benchmarks. Our PBFD formulation requires five parameters $\alpha > 1, \beta, \gamma > 0, C^{\perp}, C^{\parallel}$. In all our experiments, we set $\alpha = 2, \beta = \gamma = 1$ and only tune the last two parameters.

In our first benchmark (Figure 5), we compare the friction forces predicted using the dry friction model (Equation 1 and Equation 7), the PBD formulation (Equation 6), and our PBFD formulation (Equation 13). We simulate a 2D box sliding on the ground with an initial horizontal velocity and plot the change in sliding distance against the change in friction coefficients. The friction coefficient is μ for the dry friction model and $C^{\parallel}$ for both the PBD and PBFD formulations. Ideally, the sliding distance should consistently decrease as the friction coefficient increases. Both the dry friction model and our PBFD formulation can regenerate the ideal behavior. However, due to the underestimation of friction forces in PBD formulation, which is discussed in Section IV-B, the sliding distance erroneously increases when the friction coefficient is beyond a certain threshold. On the other hand, as we increase the timestep sizes, the simulator based on the Newton-Euler equation will become unstable but our PBFD formulation always predicts stable sliding distances. Finally, Figure 5 shows that, in order to achieve a same sliding distance, the value of $C^{\parallel}$ in PBFD formulation is quite different from the value of μ in the dry friction model. Currently we use Figure 5 to lookup $C^{\parallel}$ based on a desired sliding distance.

In our second benchmark (Figure 6 (a)), we compare our PBFD formulation and the Newton-Euler-based solver in terms of computational efficiency using three metrics. Our first metric is the timestep size (Figure 6 (b)). PBFD formulation only provides acceleration under a large timestep size. In our specific benchmark, Newton-Euler-based solver can only take $\Delta t < 0.0025$s, while our PBFD solver can take $\Delta t = 0.1$s. Under this setting, the speedup is 3X. When we use a safer timestep size for the Newton-Euler-based solver, i.e., $\Delta t = 0.001$s, the speedup is 5X. Our second metric is the average number of contact points. We observe that as the number of contact points increases, the computation time of Newton-Euler-based solver increases considerably because more complementarity conditions (Equation 7) need to be solved, while the performance of the PBFD solver is almost invariant to the number of contact points (Figure 6 (c)). Our last metric is the frictional coefficient (Figure 6 (d)). We observed that the performance of both models are almost invariant to the friction force coefficient change, although the fluctuations in the Newton-Euler solver is more evident.

Our third benchmark demonstrates the performance of

Optimizer	Objective	Substep: Compute $\frac{\partial E(\theta)}{\partial \theta}$	Substep: Compute (JTJ) Hessian	Substep: GPU Matrix Inversion	Total
LM	Equation 6	$\mathcal{O}(\|\theta\|)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$
JTJ-LM	Equation 6	$\mathcal{O}(\|\theta\|)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$
LBFGS	Equation 6	$\mathcal{O}(\|\theta\|)$	/	/	$\mathcal{O}(\|\theta\|)$
Optimizer	Objective	Substep: Compute $\frac{\partial GE(\theta)}{\partial \theta}$	Substep: Compute (JTJ) Hessian	Substep: GPU Matrix Inversion	Total
LM	Equation 13	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^3)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^3)$
JTJ-LM	Equation 13	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$	$\mathcal{O}(\|\theta\|^2)$
LBFGS	Equation 13	$\mathcal{O}(\|\theta\|^2)$	/	/	$\mathcal{O}(\|\theta\|^2)$

TABLE I: The time complexity of each iteration of optimization, including the complexity of each substep. "/" means a certain substep is not required by the particular optimizer.

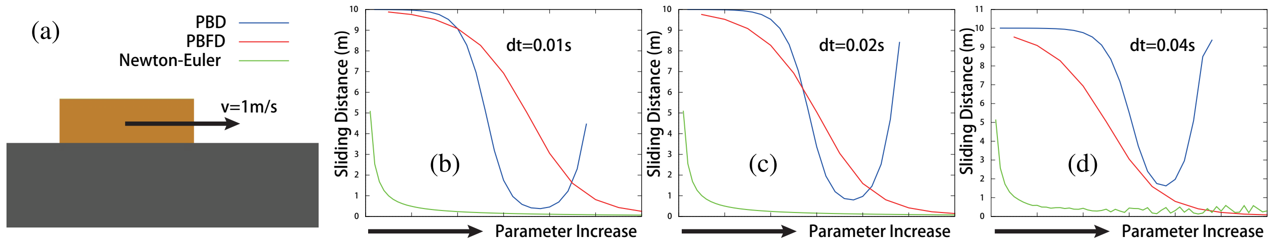


Fig. 5: (a): We simulate a box sliding on the ground with an initial velocity of 1m/s, using different formulations. We plot the box's sliding distance against the change in the parameter controlling the strength of the friction forces. The parameter is μ for the dry-friction model and $C^{\parallel}$ for the PBD and PBFD formulations. (b): In both our PBFD formulation and the dry-friction model, the sliding distance consistently decreases as the parameter increases (green, red). In the PBD formulation, however, the sliding distance increases after the parameter exceeds a certain threshold due to the underestimation of friction forces (blue). (c): As we use larger timestep sizes, the underestimation of friction forces in PBD becomes more obvious. (d): When we further increase timestep sizes, the sliding distance of the dry friction model becomes unstable (green), while our PBFD formulation is still stable.

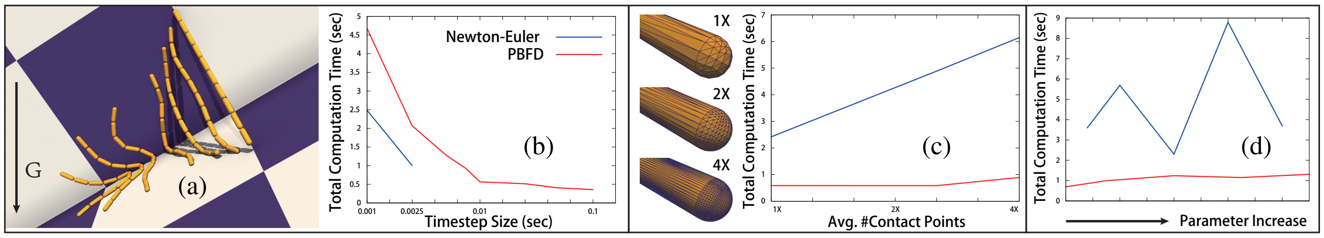


Fig. 6: (a): A 10-link chain sliding off a slope. (b): Time cost to compute a 10 seconds trajectory plotted against timestep size. For the Newton-Euler solver, the simulator is stable only when $\Delta t < 0.0025$s. (c): Time cost plotted against the average number of contact points (more contact points derived by refining the mesh). (d): Time cost plotted again the frictional coefficient (μ for the dry friction model, $C^{\parallel}$ for the PBFD formulation).

our solver in reinforcement learning applications. We select two famous RL benchmarks from [9], 2D-hopper and 3D-walker. For each benchmark, we train a neural-net controller using our PBFD formulation. We then test the neural-net controller using a conventional formulation (Equation 1 and Equation 7). Each iteration of RL takes 37 seconds for the 2D-hopper and 125 seconds for the 3D-walker on CPU. On GPU, each iteration of RL takes 6 seconds for the 2D-hopper and 31 seconds for the 3D-walker. The convergence of the RL algorithm is plotted in Figure 7, and the average reward achieved using the conventional formulation is drawn as the red bar, which is close to the best reward achieved using our PBFD formulation. This result supports our claim that the accuracy of PBFD formulation is comparable to conventional formulation on controller optimization applications.

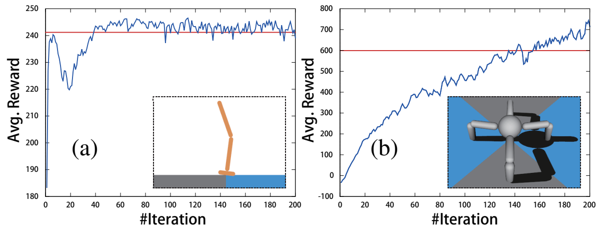


Fig. 7: We run two RL benchmarks, 2D-hopper (a) and 3D-walker (b). For each benchmark, we train a neural-net controller using our PBFD model. The convergence history of RL is plotted for each benchmark. Finally, we test the neural-net on a conventional formulation, achieving the reward shown as the red bar.

Finally, we demonstrate the speedup of GPU parallelization over the multithread CPU implementation. Our GPU implementation only provides speedup when multiple trajectories are simulated simultaneously. Therefore, in Figure 8, we plot the average computational time of GPU over CPU with respect to the number of trajectories (K), and with respect to the number of DOFs ($|\theta|$). On Nvidia Titan-X, the speedup ranges from $4-15$ times. We also observe that, as the number of simulated trajectories increases, more GPU cores are used to simulate the additional trajectories so that

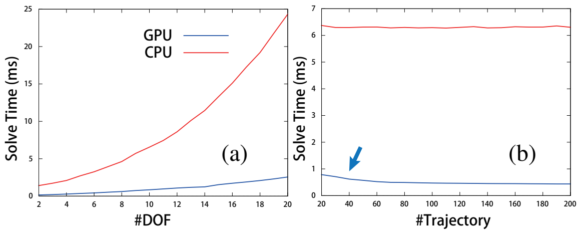


Fig. 8: (a): Average computational time to simulate one timestep on GPU/CPU plotted against the number of DOFs ($|\theta|$). The average is taken over 200 trajectories ($K = 200$), each with 1000 timesteps. (b): Average computational time to simulate one timestep on GPU/CPU plotted against the number of trajectories (K), each with 1000 timesteps. On Nvidia Titan-X, the speedup over the multithread CPU implementation ranges from $4 - 15$ times. When more trajectories are simultaneously simulated, more GPU cores are used leading to faster computation on average (blue arrow).

the average solve time for one timestep decreases, as shown in Figure 8 (b).

VII. Conclusion and Limitations

We present PBFD: a new PBD formulation that takes friction forces into account. PBFD is based on a non-integrable gradient-level PBD formation that adds friction forces as a non-integrable term. In a series of tests on practical benchmarks, PBFD performed consistently well, while PBD fails in terms of friction force estimation. We present practical algorithms to optimize the PBFD objective function and show that this algorithm can be easily parallelized on GPU to achieve $4 - 15$ times performance improvement over a multithread CPU implementation. Finally, we show that the accuracy of the PBFD solver is sufficient for certain robotic applications such as reinforcement learning by comparing the performance of a trained controller on both the PBFD formulation and the conventional formulation based on the Newton-Euler equation.

Our method has two main drawbacks. First, the entire PBD/PBFD framework is based on velocity estimation in the Euclidean space (Equation 3). However, it has been shown, e.g., [18], that respecting the Lie-group structure of the configuration space will lead to improved accuracy. Therefore, combining the PBFD formulation with Lie-group representation of velocity is a promising direction of future work. A second drawback is that our friction model is different from the dry friction model and involves five parameters that must be tuned to achieve the highest accuracy. In practice, we observe that the performance is only sensitive to the last two parameters $C^{\perp}$ and $C^{\parallel}$. This also induces future work on exploring other friction models in the PBD framework.

Acknowledgement

This research is supported in part by ARO grant W911NF16-1-0085, QNRF grant NPRP-5-995-2-415, and Intel.

References

[1] "The nvidia physx physics engine."
[2] C. T. Baker and C. A. Paul, "Computing stability regions-runge-kutta methods for delay differential equations," *IMA Journal of Numerical Analysis*, vol. 14, no. 3, pp. 347–362, 1994.
[3] J. Bender, M. Mller, and M. Macklin, "Position-Based Simulation Methods in Computer Graphics," in *EG 2015 - Tutorials*, M. Zwicker and C. Soler, Eds. The Eurographics Association, 2015.
[4] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, "Projective dynamics: Fusing constraint projections for fast simulation," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 154:1–154:11, July 2014.
[5] E. Catto, "Iterative dynamics with temporal coherence," in *Game developer conference*, vol. 2, no. 4, 2005, p. 5.
[6] E. Celledoni, H. Marthinsen, and B. Owren, "An introduction to lie group integrators–basics, new developments and applications," *Journal of Computational Physics*, vol. 257, pp. 1040–1061, 2014.
[7] C. Deul, P. Charrier, and J. Bender, "Position-based rigid body dynamics," *Computer Animation and Virtual Worlds*, vol. 27, no. 2, pp. 103–112, 2014.
[8] E. Drumwright, "A fast and stable penalty method for rigid body simulation," *IEEE transactions on visualization and computer graphics*, vol. 14, no. 1, pp. 231–240, 2008.
[9] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, pp. 1329–1338.
[10] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.
[11] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4397–4404.
[12] R. Featherstone, *Rigid Body Dynamics Algorithms*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
[13] N. Galoppo, N. K. Govindaraju, M. Henson, and D. Manocha, "Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 3.
[14] T. F. Gast, C. Schroeder, A. Stomakhin, C. Jiang, and J. M. Teran, "Optimization integrator for large time steps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 10, pp. 1103–1115, Oct 2015.
[15] F. Hahn, S. Martin, B. Thomaszewski, R. Sumner, S. Coros, and M. Gross, "Rig-space physics," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 72:1–72:8, July 2012.
[16] D. M. Kaufman, T. Edmunds, and D. K. Pai, "Fast frictional dynamics for rigid bodies," in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 946–956.
[17] D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai, "Staggered projections for frictional contact in multibody systems," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 164:1–164:11, Dec. 2008.
[18] M. Kobilarov, K. Crane, and M. Desbrun, "Lie group integrators for animation and control of vehicles," *ACM Trans. Graph.*, vol. 28, no. 2, pp. 16:1–16:14, May 2009.
[19] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 143–156.
[20] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, feb 2018.
[21] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.
[22] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, Aug 1989.
[23] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu, "Sampling-based contact-rich motion control," *ACM Trans. Graph.*, vol. 29, no. 4, July 2010.
[24] M. Macklin and M. Müller, "Position based fluids," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 104:1–104:12, July 2013.
[25] I. Mordatch, J. M. Wang, E. Todorov, and V. Koltun, "Animating human lower limbs using contact-invariant optimization," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 203:1–203:8, Nov. 2013.
[26] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.
[27] R. Narain, M. Overby, and G. E. Brown, "ADMM $\supseteq$ projective dynamics: Fast simulation of general constitutive models," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '16. Eurographics Association, 2016, pp. 21–28.
[28] Z. Pan and D. Manocha, "Articulated body dynamics simulation using optimization integrator," *arXiv preprint arXiv:1709.04145*, 2017.
[29] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2006, pp. 2219–2225.
[30] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," *arXiv preprint arXiv:1706.01905*, 2017.
[31] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction," *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.
[32] R. Tedrake and the Drake Development Team, "Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems," 2016.
[33] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 6054–6061.
[34] K. Yamane and Y. Nakamura, "A numerically robust lcp solver for simulating articulated rigid bodies in contact," *Proceedings of robotics: science and systems IV, Zurich, Switzerland*, vol. 19, p. 20, 2008.
[35] Y.-x. Yuan, "Problems on convergence of unconstrained optimization algorithms," 1998.