Exact Analytical First-Order Derivatives of MuJoCo's Smooth and Contact Dynamics for Model-Based Control

Shubham Singh

June 19, 2026

Abstract

This report derives the exact analytical first-order derivatives of a MuJoCo simulation step and demonstrates their use inside a model-predictive-control (MPC) loop. Gradient-based planners require the Jacobian of the next state with respect to the current state and control; MuJoCo computes this Jacobian by finite differencing, which re-simulates the system $\Theta(n_v)$ times per evaluation and is limited in accuracy, particularly across the non-smooth events that contact introduces. We replace it with closed-form expressions. The development proceeds in three stages. First, the smooth (contact-free) dynamics: the configuration derivative of the acceleration — the term MuJoCo evaluates only by finite difference — is expressed as a single inverse-dynamics recursion, and the two MuJoCo-specific conventions it requires (a moving reference frame and the floating base) are resolved. Second, the soft contact solve: MuJoCo's contact model is formulated as a convex optimization and differentiated exactly via the implicit function theorem, yielding a single linear system, $M + J^\top H J$, that treats sticking, slipping, and separating contacts uniformly. Third, the assembly of the full discrete transition Jacobian — the state, control, and cost/sensor Jacobians A, B, C, and D — with no finite differences anywhere in the path, a property verified by an always-on counter. The report closes with the extension to second-order (exact-Hessian) control. All results are validated to machine precision. On a trotting Unitree A1 quadruped the analytical backend computes its derivatives 10.6× faster than finite differencing (8.8× faster than the approximate WASP method) and attains a 37× lower closed-loop control cost; the cost reduction is isolated to the analytical cost-Jacobians, which eliminate a finite-difference corruption of the time-scheduled cost gradient that finite-difference-based MPC incurs implicitly.

Contents

1 Motivation and problem statement **2**

2 Background: the MuJoCo step **3**

3 Spatial rigid-body algebra **4**

4 The smooth-dynamics derivative **4**

4.1 Decomposition via inverse dynamics 4

4.2 The recursion 5

4.3 Two MuJoCo-specific corrections 5

4.4 Validation 5

5 The contact model as a convex optimization **5**

6 Differentiating the contact solve **6**
6.1 The implicit function theorem 6
6.2 A single linear system for all contact states 7
6.3 Efficient evaluation 7

7 The contact-Jacobian derivative **8**

8 Assembling the discrete transition Jacobian **8**
8.1 The cost/sensor Jacobians C and D 9

9 Empirical evaluation on a quadruped **9**

10 Extension to second order **10**

11 Summary of contributions **11**

1 Motivation and problem statement

The control problem. At each control instant, a gradient-based planner such as MPC requires the sensitivity of the next state to the current state and control. The simulator advances the state through one step,

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \tag{1}$$

where the state $\mathbf{x} = (\mathbf{q}, \mathbf{v})$ comprises the configuration $\mathbf{q}$ (joint angles and base pose) and the velocity $\mathbf{v}$. The planner consumes the **transition Jacobians**

$$A = \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} \in \mathbb{R}^{2n_v \times 2n_v}, \qquad B = \frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{u}_t} \in \mathbb{R}^{2n_v \times n_u}, \tag{2}$$

with n_v the number of velocity degrees of freedom (dofs) and n_u the number of actuators. Iterative-LQR and differential-dynamic-programming planners — the optimizers used in MuJoCo MPC [3] — evaluate A and B at every knot of the horizon and every control tick, so their computation is the dominant cost of the planning loop.

The finite-difference baseline. MuJoCo [8] computes A and B with the routine `mjd_transitionFD`, which perturbs each component of $\mathbf{x}$ and $\mathbf{u}$ by a small ϵ, re-runs the step, and forms a central difference,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\mathbf{x} + \epsilon e_i) - f(\mathbf{x} - \epsilon e_i)}{2\epsilon}. \tag{3}$$

This approach is general but incurs two costs. The first is speed: each Jacobian column requires at least one additional full simulation, so a complete Jacobian costs $\Theta(n_v)$ re-simulations, which dominates the control loop at humanoid scale ($n_v \approx 40$). The second is accuracy: central differencing is limited to approximately $\sqrt{\epsilon_{\text{mach}}} \approx 10^{-8}$, and degrades further whenever the step contains a non-smooth event — a contact activating or deactivating, or a joint reaching a limit — because the $\pm\epsilon$ probe straddles the discontinuity.

Objective. We instead compute the same Jacobians from closed-form expressions, by differentiating the equations MuJoCo solves rather than re-running the solver. This removes the re-simulation cost and attains machine precision with no ϵ trade-off. The difficulty is that a MuJoCo step is not a single expression: it consists of (i) smooth rigid-body dynamics, (ii) a contact/constraint solve, and (iii) an integration onto the configuration manifold. We differentiate each component, in Sections 4, 6–7, and 8.

Roadmap. A MuJoCo step is the composition *smooth dynamics* → *contact solve* → *integration*. We derive the exact derivative of each component and assemble them into A and B. The principal new contribution is the derivative of the contact solve; the tool that enables it is the implicit function theorem (Section 6.1).

2 Background: the MuJoCo step

Smooth forward dynamics. With contacts disabled, a rigid-body system obeys

$$M(\mathbf{q})\,\mathbf{a} = \boldsymbol{\tau} - \mathbf{c}(\mathbf{q}, \mathbf{v}), \qquad \Longrightarrow \qquad \mathbf{a} = M(\mathbf{q})^{-1}\big(\boldsymbol{\tau} - \mathbf{c}(\mathbf{q}, \mathbf{v})\big), \tag{4}$$

where $M(\mathbf{q})$ is the configuration-dependent inertia matrix, $\mathbf{a} = \ddot{\mathbf{q}}$ is the acceleration, $\boldsymbol{\tau}$ is the total applied generalized force (actuator, applied, and passive), and $\mathbf{c}(\mathbf{q}, \mathbf{v})$ — MuJoCo's `qfrc_bias` — collects the Coriolis, centrifugal, and gravitational bias forces. Equation (4) is solved internally by a factorization of M, and MuJoCo stores the resulting smooth acceleration as `qacc_smooth`.

Contacts and the constraint solve. A realistic step includes contacts. MuJoCo does not model hard impacts; it uses a soft, convex, acceleration-level constraint model. Contacts contribute a force f through the constraint Jacobian $J =$ `efc_J`, which maps joint velocities to contact-point velocities, and the constrained acceleration is

$$\mathbf{a} = \mathbf{a}_{\mathrm{s}} + M^{-1} J^{\top} f, \qquad \mathbf{a}_{\mathrm{s}} = M^{-1} \boldsymbol{\tau}_{\mathrm{smooth}}. \tag{5}$$

The force f is the solution of a small optimization, developed in Section 6. The essential point is that f is the output of an iterative solver, so $\mathbf{a}$ is defined implicitly and is not a closed-form function of $(\mathbf{q}, \mathbf{v}, \mathbf{u})$.

Integration onto the manifold. Finally, MuJoCo advances the state. With the semi-implicit Euler integrator,

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \Delta t\, \mathbf{a}, \qquad \mathbf{q}_{t+1} = \mathbf{q}_t \boxplus (\Delta t\, \mathbf{v}_{t+1}), \tag{6}$$

where $\boxplus$ denotes integration of the configuration along a velocity. For ordinary joints this reduces to $\mathbf{q}_{t+1} = \mathbf{q}_t + \Delta t\, \mathbf{v}_{t+1}$; for the floating base the orientation evolves on the rotation manifold (a unit quaternion), and $\boxplus$ applies a small rotation. This manifold structure affects the configuration block of A (Section 8).

Available and missing derivatives. MuJoCo provides one analytical derivative, `mjd_smooth_vel`, the velocity derivative of the smooth dynamics, $\partial\mathbf{a}/\partial\mathbf{v}$. The configuration derivative $\partial\mathbf{a}/\partial\mathbf{q}$, the entire contact derivative, and the manifold/integration derivative are obtained only by finite differencing. These are precisely the quantities supplied here.

3 Spatial rigid-body algebra

We differentiate rigid-body dynamics using spatial (6D) vectors, which combine the angular and linear parts of a motion or force. This section is a brief reference and may be skipped by readers familiar with Featherstone's notation [2].

A **spatial motion vector** is $\hat{v} = [\boldsymbol{\omega}; \mathbf{v}] \in \mathbb{R}^6$ (angular part first, linear part second), and a **spatial force** is $\hat{f} = [\boldsymbol{\tau}; \mathbf{f}]$. For each dof i and body b, MuJoCo maintains:

- $\texttt{cdof}_i$: the 6D motion axis of dof i (the body motion induced by that joint),
- $\texttt{cvel}_b$, $\texttt{cacc}_b$: the spatial velocity and acceleration of body b,
- $\texttt{cinert}_b$: the spatial inertia of body b.

All are expressed in world axes but referenced to the subtree center of mass $P = \texttt{subtree\_com}$. The point-Jacobian of a world point $\mathbf{p}$ attached to a body is

$$J_p(\mathbf{p})_{:,i} = \underbrace{\texttt{cdof}_i^{\text{lin}}}_{\text{linear part}} + \underbrace{\texttt{cdof}_i^{\text{ang}}}_{\text{axis}} \times (\mathbf{p} - P), \tag{7}$$

the standard "axis × lever arm" expression of screw theory.

Two products carry the analysis. The **motion cross** $\hat{v} \times_M \hat{x}$ transports a motion vector along another motion (the Lie bracket of the velocity field); the **force cross** $\hat{v} \times_F \hat{f}$ does the same for forces. Their essential property is the following.

Key identity. When the configuration is perturbed along dof k, every spatial quantity attached to a body downstream of k is transported rigidly by the motion $\texttt{cdof}_k$. To first order its change is the motion cross with $\texttt{cdof}_k$: $\partial(\text{spatial quantity})/\partial q_k = \texttt{cdof}_k \times (\text{that quantity})$.

This single identity — perturbing an upstream joint transports everything below it — generates all of the configuration derivatives in this report.

4 The smooth-dynamics derivative

We seek $\partial\mathbf{a}/\partial\mathbf{q}$ and $\partial\mathbf{a}/\partial\mathbf{v}$ for Eq. (4).

4.1 Decomposition via inverse dynamics

Differentiating $\mathbf{a} = M^{-1}(\boldsymbol{\tau} - \mathbf{c})$ directly is inconvenient because both M^{-1} and $\mathbf{c}$ depend on $\mathbf{q}$. Instead, multiply through by M and differentiate the identity $M\mathbf{a} = \boldsymbol{\tau} - \mathbf{c}$. Holding $\boldsymbol{\tau}$ fixed,

$$\frac{\partial \mathbf{a}}{\partial \mathbf{q}} = -M^{-1}\left[\frac{\partial \mathbf{c}}{\partial \mathbf{q}} + \Big(\frac{\partial M}{\partial \mathbf{q}}\Big)\mathbf{a}\right]. \tag{8}$$

The term $(\partial M/\partial\mathbf{q})\mathbf{a}$ is not negligible; omitting it discards a substantial part of the derivative. The bracket is exactly the configuration derivative of the **inverse dynamics**. Defining the inverse-dynamics map

$$\mathrm{id}(\mathbf{q}, \mathbf{v}, \mathbf{a}_0) = M(\mathbf{q})\,\mathbf{a}_0 + \mathbf{c}(\mathbf{q}, \mathbf{v}) \qquad \text{(the force producing acceleration } \mathbf{a}_0\text{)}, \tag{9}$$

we obtain

$$\boxed{\frac{\partial \mathbf{a}}{\partial \mathbf{q}} = -\,M^{-1}\,\frac{\partial\,\mathrm{id}(\mathbf{q},\mathbf{v},\mathbf{a}_0)}{\partial \mathbf{q}}\bigg|_{\mathbf{a}_0=\mathbf{a}}} \tag{10}$$

evaluated with the acceleration held fixed at the forward value $\mathbf{a}_0 = \mathbf{a}$. A single inverse-dynamics differentiation thus captures both the bias and inertia terms. The velocity derivative is simpler, since M is independent of $\mathbf{v}$: $\partial\mathbf{a}/\partial\mathbf{v} = -M^{-1}\,\partial\mathbf{c}/\partial\mathbf{v}$ (taking $\mathbf{a}_0 = 0$).

4.2 The recursion

We evaluate $\partial\,\mathrm{id}/\partial\mathbf{q}$ by one forward/backward sweep over the bodies, using the key identity of Section 3. Perturbing dof k transports the subtree below it; each spatial quantity's derivative is its motion or force cross with $\texttt{cdof}_k$, and the inertia is updated by a parallel-axis transport. Contracting the resulting force perturbations onto the dof axes yields the column $\partial\,\mathrm{id}/\partial q_k$ in $O(n_v)$ work, so the full matrix costs $O(n_v^2)$, in contrast to the $\Theta(n_v)$ re-simulations required by finite differencing; this is the same template as the analytical first-order RBD-derivative algorithms of, e.g., Pinocchio [1], specialized to MuJoCo's conventions.

4.3 Two MuJoCo-specific corrections

The recursion requires two corrections specific to MuJoCo's internal bookkeeping.

(1) The reference point moves. MuJoCo references its spatial quantities to the subtree center of mass P, which itself moves under a configuration perturbation. The simple bracket therefore captures only the reference-fixed part of the derivative, and the bare recursion is incorrect for a floating base. The remedy is to re-reference every quantity to the world origin, a genuinely fixed point: a motion vector's linear part shifts by $-\boldsymbol{\omega} \times P$ and the inertia receives a parallel-axis update, after which the transport bracket is exact and no moving-frame term is required.

(2) The floating base mixes two velocity conventions. MuJoCo's free joint stores its three translation velocities in the world frame but its three rotation velocities in the body frame; its convective term (`cdof_dot`) consequently has no clean per-dof derivative. We avoid it by recomputing the convective acceleration in body-fixed form, summing only the body-fixed dof velocities, whose derivative is a pair of motion crosses. This resolves the floating-base case.

4.4 Validation

The analytical smooth derivative matches MuJoCo's finite differences to a relative error of 10^{-8} across a range from a 9-dof arm to a 38-dof humanoid (Table 1), and is 2.6×–11.6× faster than `mjd_transitionFD`, the speedup growing with n_v (Table 2), consistent with the $O(n_v^2)$ recursion versus $\Theta(n_v)$ re-simulation scaling.

5 The contact model as a convex optimization

To differentiate the contact solve, we first state precisely what MuJoCo computes.

A soft, convex, acceleration-level model. MuJoCo does not enforce contacts as hard inequalities; it softens them. Each contact contributes a convex penalty on the constraint residual, and the

Table 1: Smooth derivative accuracy (max. relative error vs. MuJoCo FD).

robot	n_v	$\partial\mathbf{a}/\partial\mathbf{q}$	$\partial\mathbf{a}/\partial\mathbf{v}$
panda	9	5.6×10^{-9}	3.9×10^{-8}
solo12	18	5.1×10^{-8}	2.6×10^{-8}
talos	38	8.9×10^{-8}	1.1×10^{-7}

Table 2: Smooth derivative speed (μs per call).

robot	n_v	FD	analytic	speedup
panda	9	88.8	33.7	2.6×
solo12	18	238.9	45.0	5.3×
talos	38	1163.1	100.0	11.6×

constrained acceleration is the one that best balances adherence to the smooth dynamics against constraint violation. Formally, $\mathbf{a}$ minimizes

$$\min_{\mathbf{a}} \quad \tfrac{1}{2}(\mathbf{a}-\mathbf{a}_{\mathrm{s}})^{\top} M\,(\mathbf{a}-\mathbf{a}_{\mathrm{s}}) \;+\; \sum_{c} s_c(\mathbf{j}_c), \qquad \mathbf{j} = J\mathbf{a} - \texttt{aref}. \tag{11}$$

Here $\mathbf{j}$ is the **constraint residual** (`aref` is a reference acceleration encoding the desired soft restitution and stiffness), and s_c is a convex per-contact cost. The first term keeps $\mathbf{a}$ close to the unconstrained acceleration $\mathbf{a}_{\mathrm{s}}$ in the kinetic-energy metric M; the sum penalizes contact violation.

The three states of a frictional contact. For a frictional contact, s_c encodes an elliptic friction cone, whose geometry assigns each contact one of three states according to the residual:

- **Separated** (*satisfied*): the bodies are moving apart; no force, $s_c = 0$.
- **Stuck** (*quadratic*): strictly inside the cone; the penalty is a quadratic $\frac{1}{2}\mathbf{j}^{\top} D\,\mathbf{j}$ with diagonal stiffness D, so the contact behaves as a stiff spring.
- **Slipping** (*cone*): on the cone surface; the penalty is the smooth but nonlinear squared distance to the cone, and the force lies along the cone — Coulomb friction at saturation.

The force is the negative gradient of the cost, $f = -\partial s/\partial\mathbf{j}$, and the optimality condition of (11) is exactly the constrained dynamics

$$M(\mathbf{a}-\mathbf{a}_{\mathrm{s}}) - J^{\top} f = 0. \tag{12}$$

This is the equation we differentiate. A direct approach fails because f is produced by an iterative cone solver and has no closed form; the implicit function theorem resolves this.

6 Differentiating the contact solve

6.1 The implicit function theorem

Suppose a quantity y is determined by an equation $g(y,\theta) = 0$ rather than by an explicit formula, so that y is implicitly a function $y(\theta)$. The implicit function theorem (IFT) yields $dy/d\theta$ without solving for $y(\theta)$: differentiating the equation gives $\frac{\partial g}{\partial y}\frac{dy}{d\theta} + \frac{\partial g}{\partial\theta} = 0$, hence

$$\frac{dy}{d\theta} = -\Big(\frac{\partial g}{\partial y}\Big)^{-1}\frac{\partial g}{\partial\theta}. \tag{13}$$

This is the standard mechanism underlying differentiable optimization (differentiable QP/LCP layers). Here $y = \mathbf{a}$, $\theta \in \{\mathbf{q}, \mathbf{v}\}$, and g is the optimality condition (12).

6.2 A single linear system for all contact states

Let $H = \partial^2 s / \partial \mathbf{j}^2$ denote the **cost Hessian** of the contact penalty, so that $\partial f / \partial \mathbf{j} = -H$. Differentiating $g = M(\mathbf{a} - \mathbf{a}_s) - J^\top f$ at fixed $\mathbf{a}$ and combining with (13) gives

$$\big(\underbrace{M + J^\top H J}_{K_{\text{eff}}}\big)\,\frac{\partial \mathbf{a}}{\partial \theta} = M\frac{\partial \mathbf{a}_s}{\partial \theta} - \frac{\partial M}{\partial \theta}\,Wf + \frac{\partial J^\top}{\partial \theta}\,f - J^\top H\Big(\frac{\partial J}{\partial \theta}\mathbf{a} - \frac{\partial\, \texttt{aref}}{\partial \theta}\Big) + J^\top\big(f \odot g \odot \tfrac{\partial\, \text{pos}}{\partial \theta}\big), \quad (14)$$

where $W = M^{-1}J^\top$ and $Wf = \mathbf{a} - \mathbf{a}_s$. Each term has a direct interpretation:

- $K_{\text{eff}} = M + J^\top H J$ is the **effective inertia with contacts attached** — exactly the Hessian of the optimization (11). It is factored once and reused.
- $M\,\partial \mathbf{a}_s / \partial \theta$ is the smooth-dynamics derivative of Section 4, propagated through the contacts.
- $\partial M / \partial \theta\, Wf$ is the redistribution of the contact force as the inertia changes.
- $\partial J^\top / \partial \theta\, f$ and $J^\top H(\partial J / \partial \theta\, \mathbf{a} - \dots)$ are the changes in force and residual produced by the motion of the contact geometry ($J = \texttt{efc\_J}$); this is the contact-Jacobian derivative of Section 7.
- the final term $J^\top(f \odot g \odot \partial \text{pos} / \partial \theta)$ is the regularization derivative, described below.

The Hessian H encodes the contact state. The same formula covers all three states; only H changes:

$$H \;=\; \partial^2 s / \partial \mathbf{j}^2 \;=\; \begin{cases} 0, & \text{separated (no force)}, \\ D \text{ (diagonal)}, & \text{stuck (quadratic spring)}, \\ H_{\text{cone}} \text{ (dense block)}, & \text{slipping (cone surface)}. \end{cases} \quad (15)$$

The slipping block H_{cone} is the curvature of the squared-distance-to-cone cost, which MuJoCo exposes as `contact[].H`. A single factorization of $K_{\text{eff}} = M + J^\top H J$ therefore handles a scene in which some feet stick, some slip, and some separate, with no case analysis and no separate code paths. This is the central simplification.

The regularization term. MuJoCo's contact stiffness is not constant: the soft spring constant $D = 1/R$ stiffens as a contact penetrates more deeply (the impedance model). Consequently, even at a fixed residual the force f depends on the configuration through the stiffness. Differentiating this dependence yields the final term, in which $g = \texttt{impP}/(\texttt{imp}(1-\texttt{imp}))$ is the logarithmic derivative of the impedance and $\partial \text{pos} / \partial \theta$ is the penetration gradient (the normal row of J). Omitting this term is a silent error; in our implementation it appeared as a 70% error in $\partial \mathbf{a} / \partial \mathbf{q}$ that vanished once the impedance term was included.

6.3 Efficient evaluation

A direct reading of (14) would form the dense matrix $\partial M / \partial \theta\, W$ for every constraint, requiring $O(n_{\text{efc}})$ inverse-dynamics passes. The formula, however, only multiplies that matrix by f, and $\big(W^\top \partial M\, W\big) f = W^\top\big(\partial M\,(Wf)\big)$, so only the single vector $\partial M / \partial \theta\,(Wf)$ is required. This reduces the inverse-dynamics-derivative passes from $(n_{\text{efc}}{+}1)\,n_v$ to $2\,n_v$ and the per-direction cost from $O(n_{\text{efc}}^2 n_v)$ to $O(n_{\text{efc}}\,n_v)$. This contraction is what renders the analytical backend faster than finite differencing; without it, the backend is slower.

7 The contact-Jacobian derivative

The one geometric ingredient that (14) still requires is $\partial J/\partial \mathbf{q} = \partial\, \texttt{efc\_J}/\partial \mathbf{q}$, the configuration derivative of the contact Jacobian.

Definition. For a contact at world point $\mathbf{p}$ with contact frame F (one row the normal, two the friction tangents), the corresponding row of J projects the relative contact-point velocity onto a frame direction: $\texttt{efc\_J}_{r,:} = F_r \cdot J_p(\mathbf{p})$, with J_p the point-Jacobian of (7). Its configuration derivative is the kinematic Hessian of the point-Jacobian (again from the key identity: perturbing an upstream joint rotates the axes and moves the contact point), together with three contact-specific corrections.

Three contact-specific corrections.

1. **The floating base's rotation dofs transport one another.** In an ordinary chain, joint a's axis is rotated only by joints above it. The free joint's three rotation dofs, however, all move the same body and therefore mutually rotate one another's axes; the correct transport test is whether dof b moves dof a's body, not the parent ordering. The three translation axes are world-fixed and do not rotate. This coupling is the principal subtlety of the floating-base case.
2. **Geometry of the contact point.** For a sphere or capsule foot on a flat floor, the contact point reduces to a clean material point — the sphere's center, or the capsule's lower endpoint — with a factor-$\frac{1}{2}$ term on the normal direction, since the contact point lies at the penetration midpoint. This avoids any explicit collision-point derivative on the normal row.
3. **Rotation of the friction frame.** For a sphere the tangent frame is fixed; for a capsule MuJoCo aligns the friction frame with the capsule's axis, so the tangent directions rotate with the body. This introduces a $\partial F/\partial \mathbf{q}$ term on the tangent rows, with $F_1 = \text{normalize}(\mathbf{a} - (\mathbf{a}\cdot\mathbf{n})\mathbf{n})$ and $\mathbf{a}$ the axis.

Each correction is validated independently against finite differencing of $\texttt{efc\_J}$ to $\sim 10^{-11}$.

8 Assembling the discrete transition Jacobian

We now possess the continuous-time derivatives $\partial \mathbf{a}/\partial\{\mathbf{q}, \mathbf{v}\}$ (with contacts) and $\partial \mathbf{a}/\partial \mathbf{u}$, and assemble the discrete A and B of Eq. (6).

Velocity block. Since $\mathbf{v}_{t+1} = \mathbf{v} + \Delta t\, \mathbf{a}$,

$$\partial \mathbf{v}_{t+1}/\partial \mathbf{q} = \Delta t\, \partial \mathbf{a}/\partial \mathbf{q}, \qquad \partial \mathbf{v}_{t+1}/\partial \mathbf{v} = I + \Delta t\, \partial \mathbf{a}/\partial \mathbf{v}.$$

Configuration block (the manifold). For $\mathbf{q}_{t+1} = \mathbf{q} \boxplus (\Delta t\, \mathbf{v}_{t+1})$, Euclidean dofs are linear, whereas the floating-base quaternion is a manifold update. Its two Jacobians $G_q = \partial \mathbf{q}_{t+1}/\partial \mathbf{q}$ and $G_v = \partial \mathbf{q}_{t+1}/\partial \mathbf{v}_{t+1}$ are closed form: for the rotation dofs, with increment $\boldsymbol{\varphi} = \Delta t\, \mathbf{v}^{\text{rot}}_{t+1}$,

$$G_v^{\text{rot}} = \Delta t\, J_r(\boldsymbol{\varphi}), \qquad G_q^{\text{rot}} = R(\boldsymbol{\varphi})^\top, \tag{16}$$

where J_r is the right Jacobian of $SO(3)$ and R the rotation matrix of $\boldsymbol{\varphi}$ (Euclidean dofs give $G_v = \Delta t\, I$ and $G_q = I$). The configuration block of A is then $A_{qq} = G_q + G_v(\Delta t\, \partial \mathbf{a}/\partial \mathbf{q})$ and $A_{qv} = G_v(I + \Delta t\, \partial \mathbf{a}/\partial \mathbf{v})$.

Control Jacobian B. The controls enter only through the actuator force, and the contact force has no direct dependence on $\mathbf{u}$ (only through $\mathbf{a}$). The IFT therefore reduces to

$$\frac{\partial \mathbf{a}}{\partial \mathbf{u}} = K_{\text{eff}}^{-1} \frac{\partial \boldsymbol{\tau}_{\text{act}}}{\partial \mathbf{u}}, \tag{17}$$

which for affine (torque) actuators is K_{eff}^{-1} times the actuator moment arms times the gains. The discrete B then follows by the same Δt and G_v propagation as A.

8.1 The cost/sensor Jacobians C and D

The planner's running cost is a weighted sum of squared *residuals*, which MuJoCo evaluates as user sensors. The cost derivative therefore requires the residual Jacobians $C = \partial r/\partial \mathbf{x}$ and $D = \partial r/\partial \mathbf{u}$, the analogues of A, B for the sensor outputs. Two observations make these closed form.

First, the residuals are *instantaneous*: a MuJoCo step evaluates the sensors during the forward pass and then integrates without recomputing them, so C and D are the Jacobians of the residual at the current state, with no propagation through the dynamics. Most residual rows are then elementary — body, site, and geom positions and orientations (the geometric point-Jacobians of Section 3), joint-angle offsets, velocity terms, and the actuator force (which gives D directly through the actuator gains).

The one nontrivial row is the capture-point (balance) residual, which depends on the subtree center-of-mass velocity $\texttt{comvel} = J_{\text{com}}\mathbf{v}$. Its configuration derivative is a kinematic Hessian,

$$\frac{\partial\, \texttt{comvel}}{\partial q_k} = \sum_j \frac{\partial J_{\text{com}}[:, j]}{\partial q_k}\, v_j = \boldsymbol{\omega}_k \times U_{<k} \;+\; \boldsymbol{\tau}_k \times \big(\Omega_{<k} - \boldsymbol{\omega}_b\big), \tag{18}$$

assembled per body from the spatial identity $\partial \boldsymbol{\tau}_j/\partial q_k = \boldsymbol{\omega}_k \times \boldsymbol{\tau}_j$ when k is an ancestor of j and $\boldsymbol{\omega}_j \times \boldsymbol{\tau}_k$ otherwise, where $\boldsymbol{\tau}_j$ is a column of the point-Jacobian and $U_{<k}, \Omega_{<k}$ are partial linear/angular velocities accumulated from the dofs below k. We note two points of correctness. Expression (18) is *not* the time derivative $\dot{J}_{\text{com}}$, which contracts the other Hessian index; the two differ by the antisymmetric (Lie-bracket) part of the center-of-mass position Hessian, nonzero on the free joint's non-commuting rotation dofs. Accordingly, the three base-rotation dofs require the same sibling correction as the contact Jacobian. With this term, C and D match a clean central difference to machine precision.

Consequently the entire discrete transition — A, B, C, and D — is closed form, and no finite difference is evaluated on the analytical path. We verify this directly with an always-on counter that increments on any finite-difference call entering a per-knot result; it reads zero. The only remaining finite differences in the codebase are environment-gated self-checks and the validation references of Section 9.

9 Empirical evaluation on a quadruped

We integrate the analytical derivatives into MuJoCo MPC's interchangeable derivative interface and run a trotting Unitree A1, comparing three backends: finite differencing (`mjd_transitionFD`, the standard in whole-body MuJoCo MPC [5]), the approximate finite-difference-free method WASP [4], and the analytical method. The analytical backend computes all four transition blocks in closed form — the state Jacobian A, the control Jacobian B, and the cost/sensor Jacobians $C = \partial r/\partial \mathbf{x}$ and $D = \partial r/\partial \mathbf{u}$ — at every knot of the gait, spanning sphere feet, capsule shanks, sticking and slipping contacts, and joint limits. An always-on counter confirms that no finite-difference

Table 3: Closed-loop MPC on the Unitree A1 (Quadruped Flat, 250 steps, single planning thread). "model-deriv" is the time spent computing transition Jacobians; "cost" is the average per-step task cost (lower is better). The analytical backend computes derivatives an order of magnitude faster and attains a 37× lower control cost.

derivative backend	model-deriv (s)	wall-clock (s)	cost	exact?
finite difference (`mjd_transitionFD`)	127.1	186.5	0.3117	no
WASP (approximate) [4]	106.0	167.7	0.3107	no
analytic (this work)	**12.0**	**48.3**	**0.0085**	**yes**

evaluation enters the result; the only finite differences in the codebase reside in environment-gated self-checks and validation references. The comparison is therefore exact and like-for-like.

Two effects are measured. **Speed.** Computing A, B, C, D in closed form eliminates all $2n_v$ state-rollouts that finite differencing expends on the cost Jacobian, so the derivative phase is 10.6× faster than finite differencing and 8.8× faster than WASP (3.9× and 3.5×, respectively, end-to-end). Per block, the analytical transition matches a clean central difference to machine precision ($A \approx 6 \times 10^{-10}$, $B \approx 7 \times 10^{-12}$, $C \approx 2 \times 10^{-10}$, $D \approx 9 \times 10^{-12}$), whereas `mjd_transitionFD` carries an error of order ϵ from differencing the iterative solver across constraint-zone boundaries.

Accuracy and control quality. The analytical run attains a 37× lower cost (0.0085 versus 0.31). This effect is isolated rather than a confound: replacing only C, D by their finite-difference values, with A, B kept analytical, reproduces the finite-difference cost exactly (0.3106). The cause is that `mjd_transitionFD` obtains C, D by differencing through a complete `mj_step`, which re-evaluates MuJoCo MPC's time-driven gait-schedule residual; the schedule's discontinuities then appear as spurious entries of order $1/\epsilon$ (magnitude $\mathcal{O}(10^4)$) on the gait rows of C, D and corrupt the cost gradient. The analytical C, D evaluate the instantaneous residual Jacobian directly and avoid this corruption. For time-scheduled costs — a common pattern in MPC — the analytical derivative thus does not merely match finite differencing; it removes a genuine defect. We note for completeness that the end-to-end wall-clock times have each backend plan its own slightly different trajectory, whereas the per-block accuracy and the cost-isolation measurement are evaluated at identical states.

The capsule friction-frame singularity. When a capsule's axis is parallel to the contact normal, the friction tangent frame is undefined and its derivative is unbounded ($\mathrm{d}t_1/\mathrm{d}\mathbf{q} \sim 1/|u|$). This is a coordinate (gauge) singularity, not a physical one: the acceleration $\mathbf{a}$ is invariant to rotation of the tangent frame, so the unbounded term cancels in $\partial\mathbf{a}/\partial\mathbf{q}$. We therefore omit the gauge term at the singularity — verified by the observation that omitting it only near the singularity and omitting it for all capsule contacts yield identical closed-loop cost — which keeps the path finite-difference free.

10 Extension to second order

Finally, MuJoCo's tasks use an integrator that is implicit in velocity, $\mathbf{v}_{t+1} = \mathbf{v} + \Delta t(M - \Delta t\,\mathcal{D})^{-1}\boldsymbol{\tau}$, with $\mathcal{D} = \partial\boldsymbol{\tau}/\partial\mathbf{v}$ (the `mjd_smooth_vel` matrix). Expanding, $\mathbf{v}_{t+1} = \text{Euler} + \Delta t^2 M^{-1}\mathcal{D}\,\mathbf{a} + O(\Delta t^3)$, so the exact transition Jacobian contains an $O(\Delta t^2)$ correction whose derivative involves second derivatives of the bias, $\partial^2(\text{bias})/\partial\mathbf{v}^2$ and $\partial^2(\text{bias})/\partial\mathbf{q}\partial\mathbf{v}$; we verify the Δt^2 scaling numerically. The implication is that a fully exact implicit transition — and, more importantly, exact-Hessian

(Newton rather than Gauss–Newton) MPC — requires second-order rigid-body derivatives [6, 7], to which the first-order recursion of Section 4 extends by applying the same velocity sweep twice.

11 Summary of contributions

1. **The native smooth configuration-derivative** of MuJoCo's dynamics — the half it computes only by finite differencing — expressed as a single inverse-dynamics recursion, with the moving-reference and floating-base conventions resolved (Section 4).
2. **An exact derivative through MuJoCo's soft contact solve** (Section 6): a unified primal implicit-function-theorem formula whose single linear system $M + J^\top H J$ treats sticking, slipping, and separating contacts via the per-contact cost Hessian H, including the impedance/regularization term and an $O(n_{\text{efc}}) \to O(1)$ contraction that makes it efficient.
3. **A closed-form contact-Jacobian derivative** for floating-base sphere/capsule contacts (Section 7), together with the closed-form manifold and control Jacobians and the closed-form cost/sensor Jacobians C, D — including the center-of-mass-velocity kinematic-Hessian term with the free-joint sibling correction (Section 8) — so that the entire discrete transition A, B, C, D is finite-difference free, verified by a per-knot counter.
4. **A fully analytical empirical result** (Section 9): exact to machine precision, with the analytical derivatives both 10.6× faster than finite differencing (8.8× faster than WASP) and attaining a 37× lower closed-loop control cost, the latter isolated to the analytical cost-Jacobians, which eliminate a finite-difference corruption of MuJoCo MPC's time-scheduled cost gradient.

References

[1] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. 2019.

[2] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2008.

[3] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with MuJoCo, 2022. MuJoCo MPC (MJPC).

[4] Liang and Rakita. WASP: Warm-started approximate state-transition derivatives for MuJoCo. *arXiv preprint*, 2025. Approximate, finite-difference-free transition derivatives for MuJoCo.

[5] Molnar et al. Whole-body model-predictive control of legged-manipulator robots with finite-difference derivatives. *IEEE Control Systems Letters (L-CSS)*, 2025.

[6] Shubham Singh, Ryan P. Russell, and Patrick M. Wensing. Analytical second-order partial derivatives of rigid-body inverse dynamics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[7] Shubham Singh, Ryan P. Russell, and Patrick M. Wensing. On second-order derivatives of rigid-body dynamics: Theory and implementation. *IEEE Transactions on Robotics (T-RO)*, 2024.

[8] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.