Kamino: GPU-based Massively Parallel Simulation of Multi-Body Systems with Challenging Topologies

Vassilios Tsounis[1] **Guirec Maloisel**[1] **Christian Schumacher**[1] **Ruben Grandia**[1]

Agon Serifi[1] **David Müller**[1] **Chris Amevor**[2] **Tobias Widmer**[2] **Moritz Bächer**[1]

[1]Disney Research, Zurich, Switzerland [2]NVIDIA, Zurich, Switzerland

Abstract: We present Kamino, a GPU-based physics solver for massively parallel simulations of heterogeneous highly-coupled mechanical systems. Implemented in Python using NVIDIA Warp and integrated into the Newton framework, it enables the application of data-driven methods, such as large-scale reinforcement learning, to complex robotic systems that exhibit strongly coupled kinematic and dynamic constraints such as kinematic loops. The latter are often circumvented by practitioners; approximating the system topology as a kinematic tree and incorporating explicit loop-closure constraints or so-called mimic joints. Kamino aims at alleviating this burden by natively supporting these types of coupling. This capability facilitates high-throughput parallelized simulations that capture the true nature of mechanical systems that exploit closed kinematic chains for mechanical advantage. Moreover, Kamino supports heterogeneous worlds, allowing for batched simulation of structurally diverse robots on a single GPU. At its core lies a state-of-the-art constrained optimization algorithm that computes constraint forces by solving the constrained rigid multi-body forward dynamics transcribed as a nonlinear complementarity problem. This leads to high-fidelity simulations that can resolve contact dynamics without resorting to approximate models that simplify and/or convexify the problem. We demonstrate RL policy training on *DR Legs*, a biped with six nested kinematic loops, generating a feasible walking policy while simulating 4096 parallel environments on a single GPU.

Keywords: Robot Simulation, GPU-acceleration, Reinforcement Learning, Kinematic Loops

1 Introduction

GPU-based physics simulation has revolutionized robot learning in recent years. By running thousands of parallel environments entirely on device, frameworks such as Isaac Gym [1], Brax [2], and MuJoCo [3], including its GPU-accelerated variants MJX and MuJoCo Warp, reduce RL training times from days to minutes [4, 5]. The vast majority of these simulators rely on *reduced-coordinate* formulations [6] that assume a *tree-structured* kinematic topology, which enable $O(n)$ recursive algorithms with purely local data dependencies, an ideal case for GPU parallelism.

However, the tree-structured kinematic assumption excludes an important class of mechanisms: those with **kinematic loops**. Closed kinematic chains arise whenever multiple paths in the joint graph connect the same pair of bodies. They are common in parallel manipulators, legged robots with four-bar linkage transmissions [7], and hydraulic excavators [8]. These designs offer superior power-to-weight ratios and structural rigidity.

Existing simulators do support additional equality constraints that can be enforced in addition to the kinematic tree. However, reliably solving the resulting model remains a challenging task and often requires tuning of both the model and solver parameters. Closed-kinematics chains introduce unique numerical challenges as they often result in over-constrained systems and large mass-ratios. Furthermore, by introducing the somewhat arbitrary split between joints that are part of the tree versus those that are handled as equality constraints, the model is inherently biased. Finally, as they are not the main focus of existing frameworks, tooling around model specification, forward kinematics, inverse kinematics, etc., are often lacking.

Recent work has advanced both the modeling of mechanisms with kinematic loops and the solvers needed to simulate them. Schumacher et al. [9] developed inverse kinematics formulations for robots with kinematic loops, Maloisel et al. [10] proposed a quaternion-based constrained rigid body dynamics framework, and Tsounis et al. [11] provided a comprehensive treatment of forward dynamics solvers for constrained multi-body systems with loops using maximal-coordinate formulations [12]. On the solver side, proximal and ADMM-based methods for contact dynamics have matured considerably: Macklin et al. [13] introduced non-smooth Newton methods for GPU-based multi-body simulation, Tasora et al. [14] applied ADMM to variational inequalities in nonsmooth dynamics, and Carpentier et al. [15, 16] developed proximal formulations that unify compliant and rigid contacts. The theory and formulations for closed-loop simulation are in place, but a GPU implementation that brings them to RL scale has been missing.

Kamino fills this gap. It is a GPU-native solver for constrained rigid multi-body systems with arbitrary joint topologies, built on a maximal-coordinate formulation [12] where every body carries its own pose and kinematic relationships are enforced as explicit algebraic constraints. The forward dynamics problem is cast in the dual space of constraint reactions and solved via Proximal-ADMM [17, 11], naturally unifying bilateral joint constraints (including loop closures), unilateral joint limits, and frictional contacts. The central computational challenge, the linear system solve necessitated by the global coupling in the Delassus matrix introduced by kinematic loops, is addressed through either an efficient block-Cholesky decomposition (in the case of small systems) or a warm-started Conjugate Residual solver working with a block-sparse, matrix-free Delassus operator (in the case of large systems). Moreover, Kamino supports heterogeneous-worlds by design, meaning that each parallel world may include arbitrary robot morphologies. This capability opens new possibilities in batched simulation of structurally diverse robots and environments.

We demonstrate RL policy training on DR Legs [7], a biped with complex topology, including multiple kinematic loops in each leg (see Fig. 1), using up to 4096 parallel environments on a single GPU.

Kamino is implemented using NVIDIA Warp [18] and integrated into the open-source Newton physics engine [19].

Contributions.

- A GPU-accelerated physics solver to enforce hard algebraic loop-closure constraints on rigid-body systems with kinematic loops, combining a maximal-coordinate formulation with a Proximal-ADMM dual solver.
- A heterogeneous-world parallelization scheme that enables batched simulation of structurally different robots.
- Demonstrated RL policy training on DR Legs, a biped with multiple four-bar linkages per leg, as the first complex mechanism with kinematic loops trained in a GPU simulator.

2 Overview

Kamino is implemented as a solver backend in the Newton physics engine [19], built on NVIDIA Warp [18]. Newton provides the surrounding infrastructure: asset import (URDF, MJCF, USD), collision detection, and visualization. Within this framework, Kamino implements its dynamics

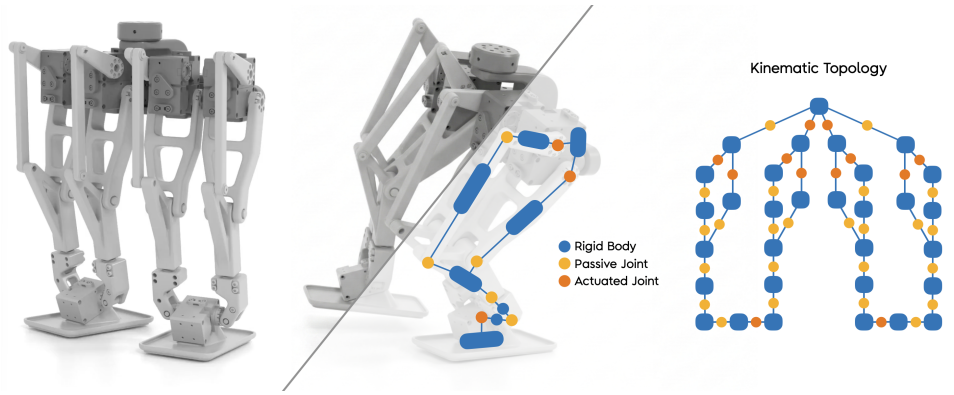


Figure 1: DR Legs (left) with kinematic graph overlaid for half of the left leg (middle). As visible in the full kinematic graph (right), each leg contains a kinematic loop, with an addition loop created between the halves of each leg. Kamino models each rigid body with an independent pose and enforces kinematic relationships, including loop closures, as explicit algebraic constraints.

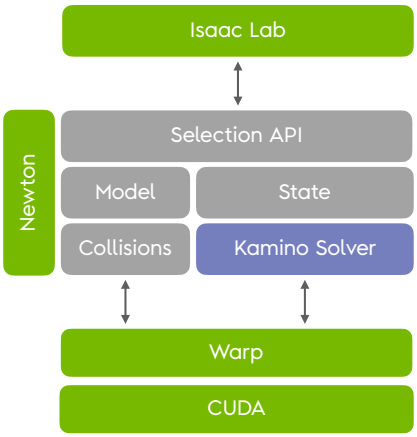


Figure 2: Kamino is a solver within Newton, which defines a common interface and core building blocks for multiple solver back-ends, all built atop of NVIDIA Warp [18]. For reinforcement learning workflows, Kamino will make use of the integration with NVIDIA's framework for robot learning, Isaac Lab, in the near future. Isaac Lab will provide environment wrappers, task definitions, and training pipelines on top of Newton's physics back-ends, enabling end-to-end RL training with Kamino.

formulation, the constraint solver, and the GPU parallelization strategy. Warp compiles Python-defined GPU kernels to CUDA and provides zero-copy tensor interoperability with PyTorch and JAX, enabling direct RL integration without host-device transfers.

The dynamics formulation and PADMM algorithm (Sec. 3, Sec. 4) follow [11]; the contributions of this paper are the GPU implementation, the engineering that makes iterative solves practical at RL rates, and the heterogeneous-world parallelization scheme (Sec. 5).

3 Constrained Dynamics in Maximal Coordinates

Kamino adopts a maximal-coordinate formulation [12] in which each rigid body i carries an independent SE(3) pose $\mathbf{q}_i = (\mathbf{r}_i, \boldsymbol{q}_i) \in \mathbb{R}^3 \times S^3$ and a 6D spatial twist $\mathbf{u}_i = (\mathbf{v}_i, \boldsymbol{\omega}_i) \in \mathbb{R}^6$. The system mass matrix is block-diagonal,

$$\mathbf{M} = \mathrm{diag}(\mathbf{M}_1, \ldots, \mathbf{M}_{n_b}), \quad \mathbf{M}_i = \begin{bmatrix} m_i \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_i \, \mathbf{I}_i^b \, \mathbf{R}_i^\top \end{bmatrix}, \tag{1}$$

where m_i is the body mass, $\mathbf{I}_i^b$ its body-frame inertia tensor, and $\mathbf{R}_i$ the rotation extracted from the quaternion $\boldsymbol{q}_i$. The equations of motion follow the Newton-Euler formulation: $\mathbf{M}\dot{\mathbf{u}} = \mathbf{h}(\mathbf{q}, \mathbf{u}) + \mathbf{J}(\mathbf{q})^\top \boldsymbol{\lambda}$, where $\mathbf{h}$ collects gravitational, gyroscopic and purely external forces, $\mathbf{J}$ is the constraint Jacobian, and $\boldsymbol{\lambda}$ the vector of constraint reactions. We refer the reader to [11] for the full derivation.

Three categories of constraints are unified in a single system:

- *Bi- and unilateral joint constraints*, $\mathbf{f}(\mathbf{q}) = 0$, enforce kinematic relationships between pairs of bodies, or a body and the world, each locking up to 6 degrees of freedom depending on the joint type. A key advantage of the maximal-coordinate formulation is that loop-closure constraints are structurally identical to ordinary joint constraints, requiring no special treatment. For actuated joints, implicit PD control is supported, where proportional-derivative gains are absorbed into the constraint Jacobian for additional numerical stability. Joint reflected inertia, often referred to as *armature*, as well as viscous joint damping, are handled similarly.
- *Unilateral joint limits* enforce inequality bounds on joint coordinates, $0 \leq \lambda \perp g(\mathbf{q}) \geq 0$.
- *Contacts* are modeled via Signorini-Coulomb-Newton complementarity with the De Saxcé correction [20, 21], which reformulates the non-associated Coulomb friction law as a complementarity condition on the Cartesian product of Coulomb cones $\mathcal{K} = \prod_k \mathcal{K}_{\mu_k}$.

The full constraint formulation, including stabilization via Baumgarte bias terms [22], follows [11].

The system is advanced in time using semi-implicit Euler integration [23]: constraint reactions are solved implicitly, and poses are then integrated explicitly via the exponential map on SO(3). A Moreau-Jean midpoint scheme [24] is also supported, which evaluates constraints at a half-step configuration for improved accuracy and stability on systems with kinematic loops.

The equations of motions together with the time-stepping scheme result in the KKT system,

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}^\top \\ \mathbf{J} & -\mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{u}^+ \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \Delta t\, \mathbf{h} + \mathbf{M}\, \mathbf{u}^- \\ -\mathbf{v}^* \end{bmatrix} , \tag{2}$$

where $\mathbf{v}^*$ collects Baumgarte stabilization, impact biases and terms arising from PD control, and $\mathbf{R}$ contains regularization terms stemming from joint-space dynamics constraints such as armature, viscous damping, and implicit PD control.

4 Proximal-ADMM Solver

After eliminating the velocity variables from the KKT system in Eq. (2) via its Schur complement, the forward dynamics problem reduces to a *dual problem in constraint reactions* [11]:

$$\boldsymbol{\lambda}^* = \underset{\mathbf{x} \in \mathcal{K}}{\operatorname{argmin}} \ \tfrac{1}{2}\, \mathbf{x}^\top \mathbf{D}\, \mathbf{x} + \mathbf{x}^\top \big(\mathbf{v}_f + \boldsymbol{\Gamma}(\mathbf{v}^+(\mathbf{x}))\big), \tag{3}$$

where $\boldsymbol{\Gamma}(\cdot)$ is the De Saxcé correction operator and

$$\mathbf{D} = \mathbf{J}\, \mathbf{M}^{-1} \mathbf{J}^\top + \mathbf{R}, \qquad \mathbf{v}_f = \mathbf{J}\big(\mathbf{u}^- + \Delta t\, \mathbf{M}^{-1} \mathbf{h}\big) + \mathbf{v}^*, \tag{4}$$

are the Delassus matrix, i.e. the constraint-space inverse apparent inertia matrix, and the free velocity vector, respectively. The De Saxcé term renders the objective nonlinear; the solver handles this by the successive approximation $\mathbf{s} = \boldsymbol{\Gamma}(\cdot)$ at the previous iterate (Alg. 1, Step 4). The biasing induced in the resulting contact forces eliminates non-zero velocities along the contact normals, ensuring that sliding contacts do not push bodies/geometries into or away from each other.

For tree-structured systems, $O(n)$ recursive algorithms such as the Articulated Body Algorithm [6] bypass the explicit assembly of $\mathbf{D}$ entirely. Kinematic loops break this structure: each loop-closure constraint couples all bodies along the loop path, producing off-diagonal blocks in $\mathbf{D}$ that prevent recursive factorization. The Delassus matrix must therefore be assembled and solved as a global system, which is the central computational challenge addressed in Sec. 5.

The dual problem Eq. (3) is a non-smooth, cone-constrained optimization problem. Following [11, 17, 15], we solve it via Proximal-ADMM (PADMM), which introduces a consensus variable $\mathbf{y}$ constrained to the feasible cone $\mathcal{K}$ and a scaled dual variable $\mathbf{z}$, together with a proximal regularization parameter η and an augmented-Lagrangian penalty ρ. The per-iteration cascade is summarized in Alg. 1. A key structural property is that the left-hand-side matrix $\mathbf{D}_{\eta,\rho} = \mathbf{D} + (\eta + \rho)\mathbf{I}$ is constant across all ADMM iterations within a timestep; only the right-hand side changes. The

Algorithm 1 Proximal-ADMM forward dynamics solver (one timestep).

Require: Delassus $\mathbf{D}$, free velocity $\mathbf{v}_f$, feasible cone $\mathcal{K}$, parameters η, ρ

Build $\mathbf{D}_{\eta,\rho} = \mathbf{D} + (\eta + \rho)\mathbf{I}$
Initialize $\mathbf{x}^0, \mathbf{y}^0, \mathbf{z}^0$ from warm-start (Sec. 4.1)
for $i = 1, \ldots, N_{\max}$ **do**
  $\mathbf{s}^i \leftarrow \boldsymbol{\Gamma}(\mathbf{z}^{i-1})$ {De Saxcé correction}
  $\mathbf{x}^i \leftarrow -\mathbf{D}_{\eta,\rho}^{-1}\big(\mathbf{v}_f + \mathbf{s}^i - \eta\,\mathbf{x}^{i-1} - \rho\,\mathbf{y}^{i-1} - \mathbf{z}^{i-1}\big)$ {linear solve (Sec. 5.1)}
  $\mathbf{y}^i \leftarrow \Pi_{\mathcal{K}}\big(\mathbf{x}^i - \rho^{-1}\mathbf{z}^{i-1}\big)$ {cone projection}
  $\mathbf{z}^i \leftarrow \mathbf{z}^{i-1} - \rho\,(\mathbf{x}^i - \mathbf{y}^i)$ {dual update}
  Apply Nesterov acceleration to $(\mathbf{y}^i, \mathbf{z}^i)$
  if $\|r_p^i\|_\infty, \|r_d^i\|_\infty, \|r_c^i\|_\infty < \epsilon$ **then**
    break
  end if
end for
return $\boldsymbol{\lambda}^* \leftarrow \mathbf{y}^N$

expensive Delassus factorization or operator setup is therefore performed *once*, and each iteration reduces to a linear solve with a new right-hand side followed by an analytical cone projection. We refer to [11] for the full derivation and convergence analysis.

4.1 Solver Features

Several techniques are combined to make the iterative solver perform well in practice.

Nesterov acceleration. We accelerate PADMM convergence with Nesterov momentum applied to the consensus and dual variables [25]. At each iteration, auxiliary variables $\hat{\mathbf{y}}^i$ and $\hat{\mathbf{z}}^i$ are computed by extrapolating from the current and previous iterates, with the momentum coefficient evolving as $a^{i+1} = (1 + \sqrt{1 + 4(a^i)^2})/2$. Adapting the restart strategy of [25], the acceleration is restarted (reset to $a = 1$) whenever the combined residual fails to decrease, preventing oscillation near the solution.

Warm-starting. Between timesteps, the converged constraint reactions $\boldsymbol{\lambda}^*$ are cached and used to initialize the next solve. For contacts, which may appear or disappear between steps, cached reactions are matched to the closest active contact by geometry pair and contact position.

Diagonal preconditioning. A diagonal Jacobi preconditioner $\mathbf{P} = \mathrm{diag}(\mathbf{D})^{-1/2}$ is computed once per timestep. The solver operates on the preconditioned system $\mathbf{PDP} + (\eta + \rho)\mathbf{I}$, which has improved spectral properties.

Practical convergence budgets. Convergence is monitored via three infinity-norm residuals (primal, dual, and complementarity), all required to drop below $\epsilon = 10^{-6}$. In practice, 10–30 PADMM iterations suffice for typical systems at $\Delta t = 1/240$ s. When using the iterative sparse linear system solver, a fixed budget of 9 iterations is used. A fixed iteration budget is preferred over adaptive tolerance because it yields deterministic kernel timing, a prerequisite for CUDA graph capture (Sec. 5).

5 GPU Parallelization

RL training requires the concurrent simulation of thousands of independent copies of the target system, each evolving in its own independent environment. Kamino addresses two GPU-specific challenges that arise in this setting: supporting *heterogeneous* simulations with constraint topology that may differ across worlds (Sec. 5.2) and efficiently solving a set of heterogeneous *linear systems* in parallel (Sec. 5.1).

5.1 Solving Linear Systems

For *global* constrained forward-dynamics solvers such as PADMM, the most significant computational bottleneck of Alg. 1 lies in the inner linear system solve (Step 5) performed at each iteration. Thus, its implementation directly determines how the solver throughput and memory footprint scales with the problem size and number of parallel worlds.

Kamino provides a few options to the user that can determine both of these aspects. As the first option, we offer two representations of the constraint Jacobian $\mathbf{J}$ matrix: a) dense or b) sparse. Second, we offer two different representations of the dynamics problem: a) a *dense* formulation that employs matrix factorization of $\mathbf{D}$, and b) a *sparse* formulation that realizes a matrix-free Delassus matrix-vector operation used in conjunction with an iterative linear system solver. The determination of which Jacobian matrix and dynamics problem representation to use is dependent on the overall system size as well as the expected density of contact constraints compared to those of joints.

The representation of the Jacobian matrix can be represented either as fully dense, or in sparse form using a Block-Sparse-Row (BSR) format, presenting significant savings both in the required memory allocations as well as the number of operations involved in its construction and use. Moreover, sparse Jacobians can be used in both dense and sparse dynamics representations, to provide additional memory savings and increased throughput even if the dynamics are solved using in dense form.

For small systems, the Delassus matrix $\mathbf{D}$ is explicitly assembled and factorized with a block-Cholesky (a.k.a. block-LLT) decomposition. Since $\mathbf{D}_{\eta,\rho}$ remains constant at each timestep (Sec. 4), a single factorization is computed before each solve and used across all PADMM iterations. Within each iteration, the cached factorization is used to perform the backward-forward passes involved in solving the linear system defined by $\mathbf{D}_{\eta,\rho}$ and the velocity bias computed from $\mathbf{v}_f$ and the decision variables of the solver.

For larger systems, the cost of assembling and factorizing a dense $\mathbf{D}$ becomes prohibitive, both w.r.t. computation time and device memory. Kamino instead stores the constraint Jacobian in block-sparse (BSR) format: since each constraint involves at most two bodies, each row has most two blocks of 6 coefficients that are non-zero. The Delassus matrix is never formed explicitly; its action $\mathbf{Dv}$ is evaluated by chaining sparse matrix-vector products with $\mathbf{J}$, $\mathbf{M}^{-1}$, and $\mathbf{J}^\top$, parallelizing over non-zero blocks. This matrix-free representation reduces memory by orders of magnitude and enables fine-grained GPU parallelism.

This matrix-free Delassus operator (computing $\mathbf{Dv}$ without forming $\mathbf{D}$) is paired with a Conjugate Residual (CR) iterative solver [13], whose per-iteration cost is dominated by the linear operator. Two GPU-specific optimizations reduce this cost:

- *Pre-baked Jacobians*: the diagonal preconditioner and $\mathbf{M}^{-1}$ are folded into a copy of the constraint Jacobian at the start of each timestep, eliminating separate scaling passes and reducing the number of memory-bound operations per CR iteration.
- *Low iteration budget*: Each solve is initialized at the previous iteration's solution, exploiting the strong correlation between successive right-hand sides as ADMM converges. As a result, a low budget of CR iterations per ADMM step is sufficient for overall convergence (as low as 9 iterations in our examples).

5.2 Heterogeneous Environments

Existing GPU simulators for RL require homogeneous environments where all worlds share the same number of simulation entities such as bodies and joints. Kamino lifts this restriction. All GPU kernels operate on 2D thread grids of shape $(\texttt{num\_worlds}, \texttt{max\_dim})$, where `max_dim` is the maximum over all worlds of the relevant quantity (bodies, constraints, contacts, etc.). Each world stores its actual dimension, and threads whose index exceeds the per-world count exit early via a mask check. Memory is allocated using the *sum* over per-world sizes (for array storage) and the *max*

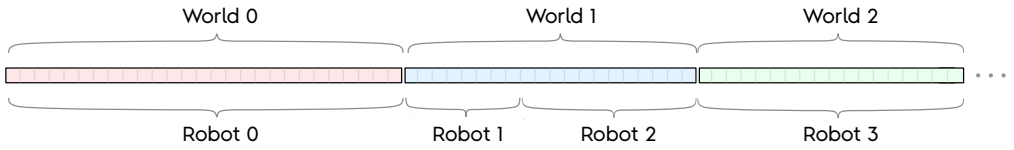


Figure 3: Kamino supports heterogeneous worlds. Each world may contain a different robot or set of robots with a variable amount of bodies and joints.

(for thread grid dimensions). A Struct-of-Arrays layout ensures coalesced GPU memory access: threads in the same warp access consecutive memory locations across worlds.

To eliminate kernel launch overhead, the entire PADMM solver loop (Alg. 1) is captured as a CUDA graph using Warp's `wp.capture_while`, which compiles the iterative loop into a single graph with conditional re-execution. This is possible because all array shapes are fixed at graph capture time and the Delassus operator is constant within a timestep. Additionally, a per-world mask allows early exit for worlds in which PADMM has already converged. The captured graph is replayed each timestep, amortizing the construction cost over the training run.

6 Experiments

6.1 Systems

We evaluate Kamino on four robotic systems of varying complexity (Fig. 4, Tab. 1), covering both tree-structured and closed-loop topologies.

- **DR Legs** [7] is a bipedal robot with serial-parallel hybrid legs containing multiple four-bar linkages (Fig. 1). Each of the four half-legs (left/right $\times$ inner/outer) contains 9 revolute joints, yielding 36 joints total (12 actuated, 24 passive) with6 independent kinematic loops.
- **BDX** is a 14-DOF bipedal humanoid with a tree-structured topology.
- **Olaf** is the robotic representation of a well known character with 25 actuated DOFs spanning legs, torso, neck, arms, and facial features. Five kinematic loops arise from mechanical couplers in the jaw, shoulders, eye pitch, and eyelid mechanisms.
- **Iron Man** is a 27-DOF Audio-Animatronics® figure fixed to the ground with a complex topology, including 45 kinematic loops.

(a) DR Legs (b) BDX (c) Olaf (d) Iron Man © MARVEL, Disneyland Paris

Figure 4: Robotic systems simulated with Kamino. DR Legs and Olaf contain closed kinematic chains; BDX and Iron Man are tree-structured.

Table 1: **Systems simulated with Kamino.** All quantities refer to a single instance in maximal coordinates. Loops denote independent closed kinematic chains.

System	Bodies	Actuated	Passive	Loops
DR Legs	31	12	24	6
BDX	21	14	0	0
Olaf	36	25	15	5
Iron Man	151	27	169	45

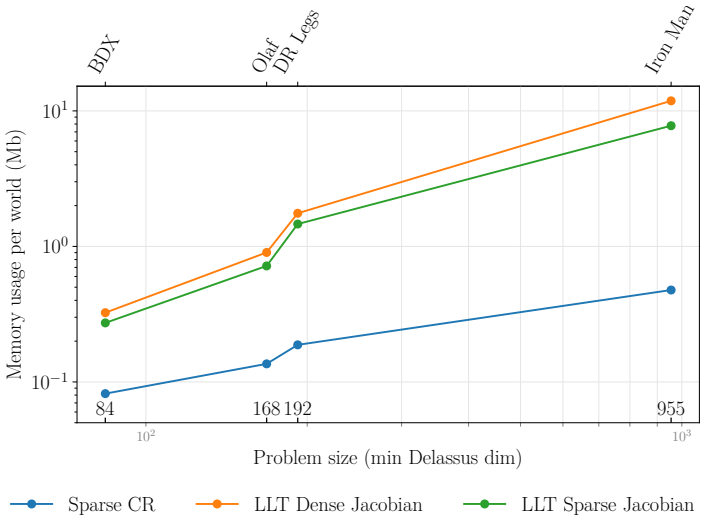


Figure 5: **Memory usage for sparse vs dense solvers.** Per-world memory usage in Mb for each problem and solver, against the problem size.

6.2 Simulation Performance

To evaluate the time and memory performance of our simulator, we consider three solver configurations: one using the sparse CR solver and two using the dense LLT solver, with either a dense or sparse representation of the constraint Jacobian.

Since the size of a problem (e.g., the number of rows of the Delassus matrix) varies through time, as new contacts open and close, we pick the minimal size of the Delassus matrix (that is, when no contact or limit constraints are active) to represent the problem size when evaluating the scaling of our solvers. This is reasonable for scenarios where the number of contacts remains low (e.g., walking).

Memory Usage For a given problem and solver configuration, the memory footprint of the simulator is the sum of a constant term (made up largely of the robot's geometry, and independent of the solver configuration) and of a variable term that scales linearly with the number of parallel worlds. As depicted in Fig. 5, we measure this per-world cost for each robot and solver configuration, and plot it against the problem size.

As expected, our sparse solver has a significantly smaller memory footprint than the dense solvers. More specifically, the memory cost scales quadratically with the problem size for dense solvers (dominated by the storage of the Delassus matrix, and possibly the dense Jacobian), but only linearly if a sparse representation is used (since every constraint involves at most two bodies, there is a fixed maximal number of non-zero coefficients per row in the constraints Jacobian).

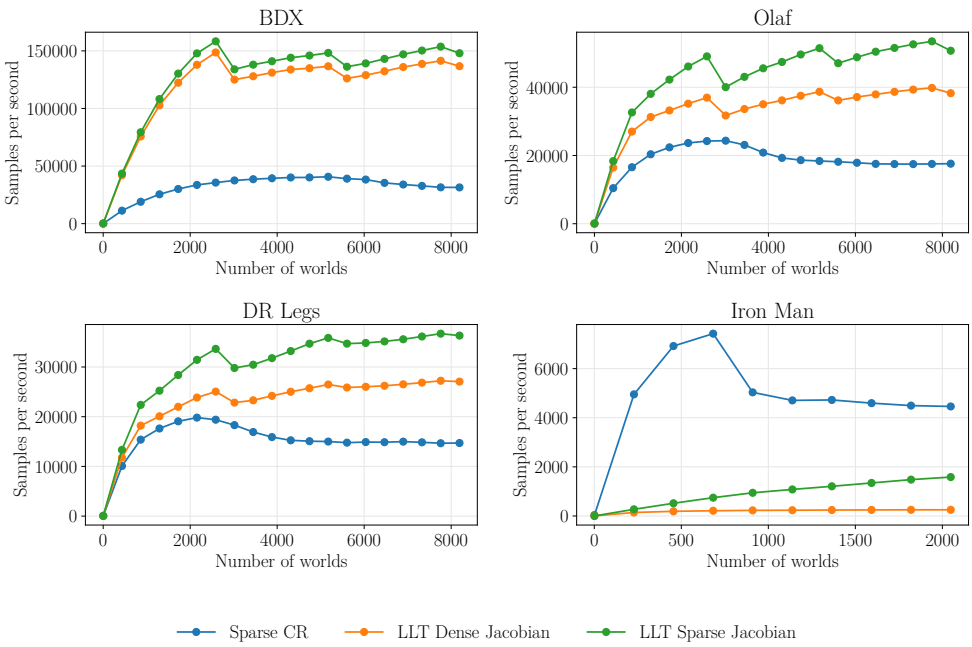


Figure 6: **Simulator throughput vs number of parallel worlds.** We measure, for every example and solver, and for varying numbers of parallel worlds, the total number of simulator samples (number of steps per second, times number of worlds). Experiments were run on a RTX Pro 6000 GPU.

Throughput With RL applications in mind, the time performance of our simulator is evaluated in terms of its maximal throughput, that is, the number of simulator steps per second, multiplied by the number of parallel worlds.

Fig. 6 illustrates, for individual examples, the simulator throughput for varying numbers of worlds. Unsurprisingly, dense solvers are more efficient for small problems, but become prohibitive for complex robots such as Iron Man. It is also apparent that using a sparse data structure for assembling the Delassus matrix is beneficial for all problems.

In all cases, the throughput stops increasing beyond a few thousand worlds, as arithmetic units saturate. The throughput even decreases slightly beyond its peak for sparse solvers, presumably as cache efficiency decreases, as a consequence of the irregular memory accesses that come with sparsity. Note that the sawtooth patterns in the curves corresponding to the dense solvers are produced by the tail effect: the throughput drops when the number of scheduled threads just exceeds the SM capacity, requiring an additional execution wave.

Extracting the maximum throughput observed for individual examples, and plotting it against the problem size in Fig. 7, we can more clearly characterize the scaling of the 3 solver options with the problem size. Additionally, we can identify the point at which our sparse solver becomes faster than the dense variants, that is, for about 300 individual constraint equations.

Hardware To validate that our results translate to multiple hardware configurations, and to provide a rough estimate of the associated speedup factor, we run our throughput profiling for the DR Legs example on several GPU models, namely a RTX 4090, a RTX 5090 and a RTX Pro 6000 graphics cards, as displayed in Fig. 8.

6.3 Reinforcement Learning

We train RL policies on all walking systems using PPO [26] with massively parallel GPU environments and using the RSL-RL training framework [4, 27].

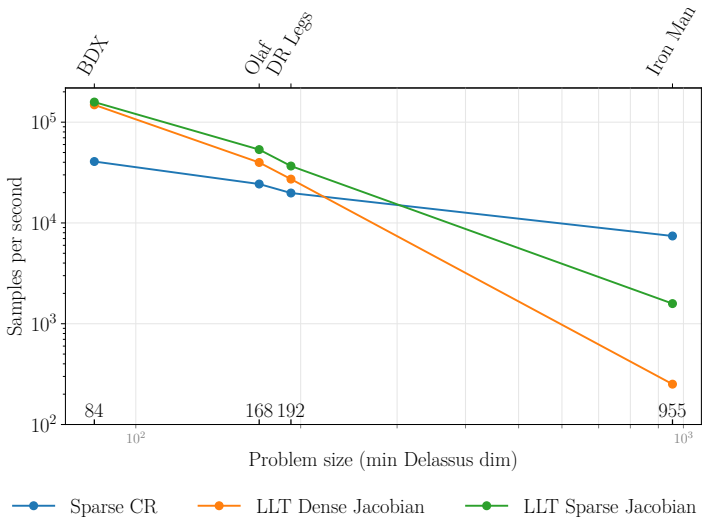


Figure 7: **Simulator maximal throughput vs problem size.** The maximal throughput from the experiments of Fig. 6, for each problem and solver configuration.

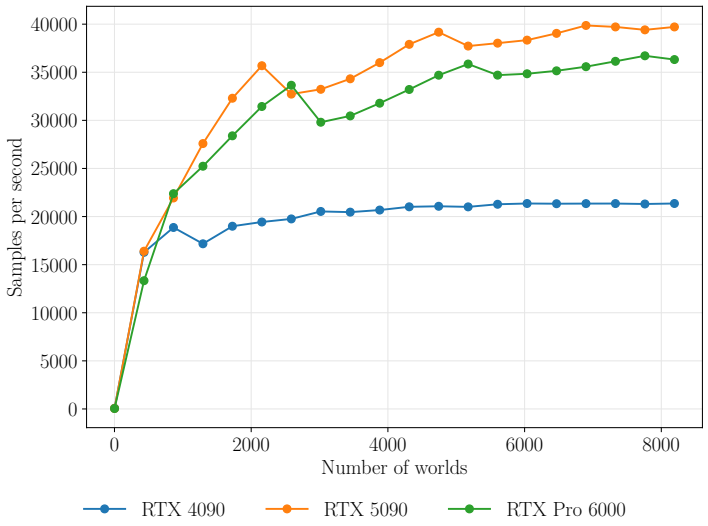
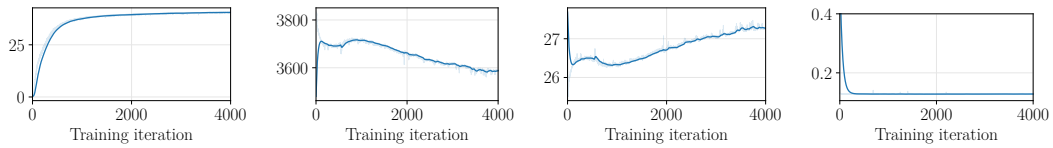


Figure 8: **Throughput vs number of worlds for different hardware models.** We reproduce the experiments of Fig. 6 on different GPU models, for the DR Legs example with the most appropriate solver configuration (i.e., dense LLT with sparse Jacobian).

DR Legs Kamino enables, for the first time, GPU-accelerated RL training of a robot with native handling of complex kinematic loops. The full system with all 6 kinematic loops and 12 actuated and 24 passive joints is trained end-to-end. The simulation uses the Moreau-Jean midpoint integrator at 250 Hz and the PADMM solver with a sparse Jacobian and matrix-free Conjugate Residual linear solver, warm-starting, and Nesterov acceleration. We simulate 4096 parallel environments with implicit PD control (K_p=15 N m rad^{-1}, K_d=0.6 N m s rad^{-1}) and a policy rate of 50 Hz (decimation of 5). The policy is a 3-layer MLP (512×512×512, ELU activations) mapping a 94-dimensional observation (root orientation, velocity, gait phase, joint positions, and action history) to 12 normalized joint position targets. The reward combines velocity and heading tracking, gait contact matching, foot clearance, root orientation, feet parallelism, a survival bonus, and regularization penalties on torques, action rate, vertical/angular velocity, and foot touchdown impacts. Episodes terminate on



(a) Mean episode reward (b) Total FPS [env steps/s] (c) Collection time [s] (d) Learning time [s]

Figure 9: DR Legs training with 4096 parallel environments on a single RTX Pro 6000 GPU.

pelvis ground contact or feet self-collision. Training runs for 4000 PPO iterations ($\sim$390M environment transitions) at roughly 3600 environment steps per second, converging in about 31 h of wall-clock time on a single GPU. Fig. 9 shows the reward curve, simulation throughput, and the per-iteration breakdown into environment collection time and PPO learning time.

Olaf We simulate the full system, including the closed-loop mechanisms in the jaw and shoulder couplers, to demonstrate that Kamino can faithfully reproduce the dynamics of complex characters with mixed tree and loop topologies. See video.

BDX We verified successful sim-to-sim transfer of RL policies trained in Kamino to Isaac Sim and from Isaac Sim to Kamino, validating simulation fidelity.

Forward kinematics for environment resets. Resetting environments with kinematic loops requires setting not only the actuated joint positions but the full system state, including all passive joints. To this end, we implemented an efficient forward kinematics (FK) solver, using the Gauss-Newton method to computes consistent body poses from a given set of joint coordinates.

We used this FK capability to simulate Iron Man, a complex Audio-Animatronics® figure, demonstrating that Kamino can handle systems beyond those used for RL training.

The integration of this solver in the RL training pipeline, to allow resetting robots in parallel worlds to poses with randomized joint angles, remains to be addressed in future work, as for instance it remains challenging to ensure that the sampled poses are free of self-collisions or kinematic singularities.

7 Discussion

The results presented in the previous section provide experimental validation of Kamino's capabilities. First we verify that our solver can provide massively parallel simulations of complex robots with high fidelity, but at a relatively lower throughput compared to the reduced-coordinate solvers available within Newton. Second, we demonstrate that Kamino provides an effective back-end for applying reinforcement learning to such challenging mechanical systems.

It is important to emphasize the trade-offs of adopting a maximal-coordinate formulation. For simple articulated systems such as BDX, resolving the dynamics of bilateral joint constraints explicitly, incurs a significant overhead as the overall dimensionality of the dynamics increases compared to reduced-coordinate solvers. However, it handles robots with arbitrary topology in a unified manner without requiring users to manually define the base kinematic tree. At the scale of systems such as Iron Man with over 200 joints, such an undertaking can prove especially challenging.

In terms of limitations, it is currently targeted exclusively for rigid multi-body systems, i.e., particles and soft bodies are not yet supported, and it does not provide end-to-end solver differentiability for use in gradient-based optimization. However, all of the aforementioned features are part of ongoing and future work. Moreover, the current implementation is limited to only employing the dual formulation of the forward dynamics for resolving constraint reactions. We are, however, also working towards supporting a *primal* and *KKT* variants of the inner linear-system. The former allows the forward-dynamics problem to remain dimensionally invariant to the number of constraints,

and may therefore provide better scaling for worlds involving large number of contacts compared to the number of bodies. The KKT system, on the other hand, can provide advantages for systems with excessively sparse couplings since parallelized matrix-vector products can fully exploit SIMT parallelism on the GPU. Moreover, compared to primal and dual Schur complement variants, avoiding the compounded scaling effects of body inertia and constraint lever-arms, can reduce the dependence on preconditioning for ensuring numerical stability at 32-bit floating-point precision.

Lastly, we remark on on how our experiments have shown successful sim-to-sim transfer of RL policies. An important next step for Kamino is to demonstrate sim-to-real transfer onto a physical robot to validate that our solver can simulate physical phenomena with sufficient realism. This is particularly important w.r.t. sufficiently capturing actuator dynamics as well as the behavior of frictional contacts and restitutive impacts. These aspects are often the dominant sources of the so-called *reality gap* between simulated systems and their physical counterparts [28].

8 Conclusion

We presented a GPU-based physics solver for massively parallel simulations of heterogeneous and highly-coupled mechanical systems. Our solver enables robotic system developers to fully exploit the mechanical advantage afforded by kinematic loops, without requiring them to simplify the system representation for simulation and RL policy training. In addition, the ability to handle heterogeneous worlds, enables the massively parallel simulation of structurally diverse robots, opening the door for employing sampling-based techniques for optimizing the mechanism design parameters. Our experimental validation has demonstrated that Kamino performs best when simulating thousands of parallel worlds, which fully exploits the GPU's SIMT parallelism. The cost in throughput incurred by the use of a maximal-coordinate formulation is thus effectively mitigated by ability to simulated a very large number of parallel instances of the target system. This capability facilitates large-scale reinforcement learning on complex mechanisms, such as those presented in this work.

Acknowledgments

We thank Josefine Klintberg and Espen Knoop at Disney Research for their contributions and feedback, Gilles Daviet, Eric Heiden, Miles Macklin, Gavriel State, Alain Denzler, Philipp Reist, Adam Moravanszky, and Spencer Huang at NVIDIA for their collaboration on the Newton framework and code reviews, and Google DeepMind for insightful discussions throughout the development of this work.

References

[1] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu based physics simulation for robot learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[2] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax – a differentiable physics engine for large scale rigid body simulation. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2021.

[3] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi:10.1109/IROS.2012.6386109.

[4] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 91–100, 2022.

[5] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi:10.1109/LRA.2023.3270034.

[6] R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer, 2014.

[7] K. G. Gim, J. Kim, and K. Yamane. Design and fabrication of a bipedal robot using serial-parallel hybrid leg mechanism. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5095–5100, 2018. doi:10.1109/IROS.2018.8594182.

[8] P. Egli and M. Hutter. A general approach for the automation of hydraulic excavator arms using reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):5679–5686, 2022. doi:10.1109/LRA.2022.3152865.

[9] C. Schumacher, E. Knoop, and M. Bächer. A versatile inverse kinematics formulation for retargeting motions onto robots with kinematic loops. *IEEE Robotics and Automation Letters*, 6(2):943–950, 2021.

[10] G. Maloisel, C. Schumacher, E. Knoop, R. Grandia, and M. Bächer. A versatile quaternion-based constrained rigid body dynamics. *ACM Transactions on Graphics (SIGGRAPH)*, 2025.

[11] V. Tsounis, R. Grandia, and M. Bächer. On solving the dynamics of constrained rigid multi-body systems with kinematic loops. *arXiv preprint arXiv:2504.19771*, 2025.

[12] E. J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems*. Allyn and Bacon, 1989.

[13] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and V. Makoviychuk. Non-smooth newton methods for deformable multi-body dynamics. *ACM Transactions on Graphics*, 38(5):1–20, 2019.

[14] A. Tasora, D. Mangoni, S. Benatti, and R. Garziera. Solving variational inequalities and cone complementarity problems in nonsmooth dynamics using the alternating direction method of multipliers. *International Journal for Numerical Methods in Engineering*, 122(16):4093–4113, 2021. doi:10.1002/nme.6693.

[15] J. Carpentier, R. Budhiraja, and N. Mansard. Proximal and sparse resolution of constrained dynamic equations. In *Robotics: Science and Systems*, 2021.

[16] J. Carpentier, Q. Le Lidec, and L. Montaut. From compliant to rigid contact simulation: a unified and efficient approach. In *Robotics: Science and Systems*, 2024.

[17] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011. doi:10.1561/2200000016.

[18] M. Macklin. Warp: A high-performance python framework for gpu simulation and graphics. `https://github.com/nvidia/warp`, March 2022. NVIDIA GPU Technology Conference (GTC).

[19] Newton Contributors. Newton: GPU-accelerated physics simulation for robotics and simulation research. `https://github.com/newton-physics/newton`, 2025. Newton, a Series of LF Projects, LLC, Apache-2.0 License.

[20] G. de Saxcé and Z.-Q. Feng. The bipotential method: A constructive approach to design the complete contact law with friction and improved numerical algorithms. *Mathematical and Computer Modelling*, 28(4):225–245, 1998.

[21] V. Acary, F. Cadoux, C. Lemaréchal, and J. Malick. A formulation of the linear discrete coulomb friction problem via convex optimization. *ZAMM – Journal of Applied Mathematics and Mechanics*, 91(2):155–175, 2011.

[22] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16, 1972.

[23] 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*, 39:2673–2691, 1996.

[24] M. Jean. The non-smooth contact dynamics method. *Computer Methods in Applied Mechanics and Engineering*, 177(3-4):235–257, 1999. doi:10.1016/S0045-7825(98)00383-1.

[25] B. O'Donoghue and E. Candès. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3):715–732, 2015.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[27] C. Schwarke, M. Mittal, N. Rudin, D. Hoeller, and M. Hutter. Rsl-rl: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025.

[28] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science robotics*, 4(26):eaau5872, 2019.