Robotics: Science and Systems 2022
New York City, NY, USA, June 27-July 1, 2022

Factory: Fast Contact for Robotic Assembly

Yashraj Narang*, Kier Storey*, Iretiayo Akinola*, Miles Macklin*, Philipp Reist*, Lukasz Wawrzyniak*, Yunrong Guo*, Adam Moravanszky*, Gavriel State*, Michelle Lu*, Ankur Handa*, Dieter Fox*†
*NVIDIA Corporation, Santa Clara, USA
†Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, USA

***Abstract*—Robotic assembly is one of the oldest and most challenging applications of robotics. In other areas of robotics, such as perception and grasping, simulation has rapidly accelerated research progress, particularly when combined with modern deep learning. However, accurately, efficiently, and robustly simulating the range of contact-rich interactions in assembly remains a longstanding challenge. In this work, we present *Factory*, a set of physics simulation methods and robot learning tools for such applications. We achieve real-time or faster simulation of a wide range of contact-rich scenes, including simultaneous simulation of 1000 nut-and-bolt interactions. We provide 60 carefully-designed part models, 3 robotic assembly environments, and 7 robot controllers for training and testing virtual robots. Finally, we train and evaluate proof-of-concept reinforcement learning policies for nut-and-bolt assembly. We aim for *Factory* to open the doors to using simulation for robotic assembly, as well as many other contact-rich applications in robotics. Please see our website for supplementary content, including videos.** [1]

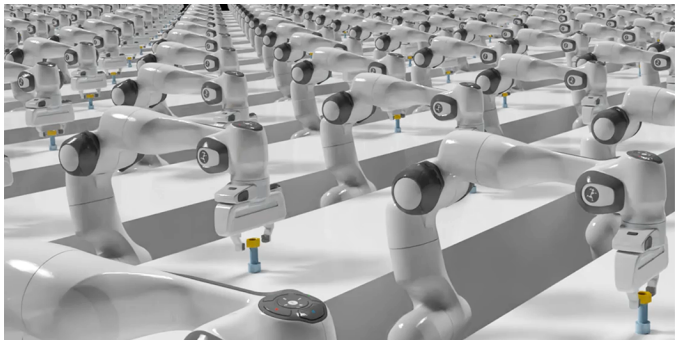

Fig. 1. Rendering of Franka robots interacting with nut-and-bolt assemblies in Isaac Gym using methods from **Factory**. The simulation contains 128 parallel environments and is executing in real-time on a single GPU.

I. Introduction

Assembly is an essential, but highly challenging area of manufacturing. It includes a diverse range of operations, from peg insertion, electrical connector insertion, and threaded fastener mating (*e.g.*, tightening nuts and bolts), to wire processing, cable routing, and soldering [9, 41]. These operations are ubiquitous across the automotive, aerospace, electronics, and medical industries [9]. However, assembly has been exceptionally difficult to automate due to physical complexity, part variability, and strict reliability requirements [41].

In industry, robotic assembly methods may achieve high precision, accuracy, and reliability [9, 51, 97]. However, these methods can be highly restrictive. They often use expensive equipment, require custom fixtures, have high setup times (*e.g.*, tooling design, waypoint definition, parameter tuning) and cycle times, and are sensitive to variation (*e.g.*, part appearance, location). Custom tooling and part-specific engineering are also cost-prohibitive for high-mix, low-volume settings [41]. In research, methods for robotic assembly often use less-expensive equipment, require fewer custom fixtures, achieve increased robustness to variation, and may recover from failure [34, 55, 89]. Nevertheless, these methods often have lower reliability, higher setup times (*e.g.*, demo collection, real-world training, parameter tuning), and/or higher cycle times.

Meanwhile, physics simulation has become a powerful tool for robotics development. Simulators have primarily been used to verify and validate robot designs and algorithms [1]. Recent research has demonstrated a host of other applications: creating training environments for virtual robots [19, 49, 90, 104], generating large-scale grasping datasets [20, 32], inferring real-world material parameters [69, 70], simulating tactile sensors [72, 83, 99], and training reinforcement learning (RL) agents for manipulation and locomotion [13, 77]. Compelling works have now shown that RL policies trained in simulation can be transferred to the real world [2, 4, 12, 26, 61, 80, 102].

Nevertheless, the power of physics simulation has not substantially impacted robotic assembly. For assembly, a simulator must accurately and efficiently simulate contact-rich interactions, a longstanding challenge in robotics [14, 30, 52, 108], particularly for geometrically-complex, tight-clearance bodies. For instance, consider the canonical nut-and-bolt assembly task. Real-world nuts and bolts have finite clearances between their threads, thus experiencing 6-DOF relative motion rather than pure helical motion. To simulate real-world motion phases (*e.g.*, initial mating, rundown) and associated pathologies (*e.g.*, cross-threading, jamming) [35], collisions between the threads must be simulated. However, high-quality surface meshes for a nut-and-bolt may consist of $10k$-$50k$ triangles; a naive collision scheme may easily exceed memory and compute limits. Moreover, for RL training, a numerical solver may need to satisfy non-penetration constraints for 1000 environments in real-time (*i.e.*, at the same rate as the underlying physical dynamics). Despite the omnipresence of threaded fasteners in the world, no existing simulator achieves this performance.

In this work, we present **Factory**, a set of physics simulation methods and robot learning tools for such interactions (**Fig. 1**). Specifically, we contribute the following:

[1] https://sites.google.com/nvidia.com/factory/

- **A physics simulation module** for fast, accurate simulations of contact-rich interactions through a novel synthesis of signed distance function (SDF)-based collisions, contact reduction, and a Gauss-Seidel solver. The module is accessible within the PhysX physics engine [74] and Isaac Gym [63]. We demonstrate simulator performance on a wide range of challenging scenes. As an example, we simulate 1000 simultaneous nut-and-bolt assemblies in real-time on a single GPU, whereas the prior state-of-the-art was a single nut-and-bolt assembly at $\frac{1}{20}$ real-time.
- **A robot learning suite** consisting of a Franka robot and all rigid-body assemblies from the NIST Assembly Task Board 1 [41], the established benchmark for robotic assembly [97]. The suite includes 60 carefully-designed assets, 3 robotic assembly environments, and 7 classical robot controllers. The suite is accessible within Isaac Gym. User-defined assets, controllers, and environments can be added and simulated as desired.
- **Proof-of-concept RL policies** for a simulated Franka robot to solve the most contact-rich task on the NIST board, nut-and-bolt assembly. The policies are developed within Isaac Gym. The contact forces generated during policy execution are compared to literature values from the real world and show strong consistency.

We aim for **Factory** to greatly accelerate research and development in robotic assembly, as well as serve as a powerful tool for contact-rich simulation of any kind.

II. Related Work

A. Contact-Rich Simulation

Contacts are sets of points on shape pairs that represent currently (or potentially) overlapping parts where forces may be applied. A longstanding challenge in contact-rich simulation is fast, accurate, and robust contact generation, as well as solution of non-penetration constraints. We have found that achieving such performance requires careful consideration of 1) geometric representations, 2) contact reduction schemes, and 3) numerical solvers. Here we review primary options for each, as well as prior results on a challenging benchmark.

1) Geometric Representations: There are 5 major geometric representations in physics simulation for graphics: convex hulls, convex decompositions, triangular meshes (trimeshes), tetrahedral meshes (tetmeshes), and SDFs (**Fig. S10**).

Convex hulls cannot accurately approximate complex object geometries, such as threaded fasteners with concavities. **Convex decompositions** address this issue by approximating the input shape using multiple convex hulls, generated with algorithms such as V-HACD [64]. While an improvement on single convex hulls, these decompositions can produce spatial artifacts on complex geometries (**Fig. S11**). Even for perfect decompositions, the number of collision pairs to test during contact generation scales as $\mathcal{O}(n^2)$ (where n is the number of convex shapes), impacting memory and performance. Since contacts are generated between convex shapes, undesirable contact normals can be generated, and snagging may occur.

Trimeshes can provide a near-exact approximation of complex geometries. However, the number of collision pairs for contact generation scales as $\mathcal{O}(n^2)$ (where n is the number of triangles), again impacting memory and performance. In addition, since triangles have zero volume, penetrations can be difficult to resolve, motivating techniques such as boundary-layer expanded meshes [27]. **Tetmeshes** can mitigate such penetration issues, but high quality tetrahedral meshing is challenging. Tetrahedra may have extreme aspect ratios in high-detail areas, leading to inaccurate collision checks.

SDFs, which map points to distance-to-a-surface, can provide accurate implicit representations of complex geometries. They enable fast lookups of distances, as well as efficient computation of gradients to define contact normals. However, using SDFs for collisions requires precomputing SDFs offline from a mesh, which can be time- and memory-intensive. Moreover, collision schemes typically test the vertices of a trimesh against the SDF to generate contacts. For sharp objects, simply sampling vertices can cause penetration to occur, motivating iterative per-triangle contact generation [60].

We use discrete, voxel-based SDFs as our geometric representation and demonstrate that they provide efficient, robust collision detection for challenging assets in robotic assembly.

2) Contact Reduction: Contact reduction is a powerful technique from game physics for reducing the total number of contacts without compromising simulation accuracy. A naive contact generation scheme between a nut and bolt may generate $\sim$1e6 contacts; a careful one may generate $\sim$1e4. Excessive contacts can impact both memory and stability, and per-contact memory requirements cannot easily be reduced. Since the rigid-body mechanics principle of equivalent systems dictates that a set of distributed forces can be replaced by a smaller set of forces (constrained to produce the same net force and moment), accurate dynamics can be preserved.

Specifically, contact reduction consists of methods for preprocessing, clustering, and maintaining temporal persistence of contacts. We pay particular attention to **contact clustering**, which generates bins for contacts, reduces the bins, and reduces the contacts within each bin using heuristics. The two most common heuristics are normal similarity and penetration depth [71]. **Normal similarity** assigns contacts with similar surface normals to the same bin, culls similar bins, and culls similar contacts, and is often implemented with k-means or cube map algorithms [75, 79, 25]. **Penetration depth** culls bins and contacts with negligible penetration, as these often have minimal impact on dynamics. **Data-driven** methods train networks to perform the reduction [40], but require a separate data collection and learning phase to be effective.

We combine normal similarity, penetration depth, and an area-based metric to reduce contacts and demonstrate dynamics across numerous evaluation scenes.

3) Solvers: There are 4 major options for solvers in physics simulation: direct, conjugate gradient/conjugate residual (CG/CR), Jacobi, and Gauss-Seidel. We briefly review these solution methods below; see [3] for a detailed treatment.

Direct solvers, which may rely on matrix pivoting, do not

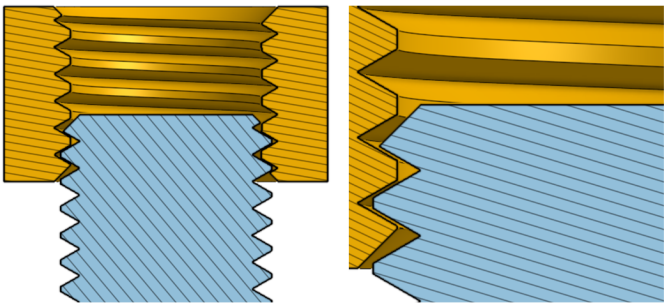

Fig. 2. Cross-sectional view of an aligned M16 nut-and-bolt assembly. Nuts and bolts have finite clearances between their threads, thus experiencing 6-DOF kinematics with multi-surface frictional contact, rather than pure helical motion. As follows, a nut can roll and pitch on a bolt, and may experience pathologies such as cross-threading and jamming [35]. Left: a *loose* model, where the clearances between threads are at their maximum allowable value (ISO 965). Right: zoom view. Thread roots are not modeled.

scale well to large sets of contact constraints, such as the $\sim$1e4 constraints between a nut and bolt mesh. **CG/CR methods** can handle larger sets of constraints, but are still unable to achieve real-time performance for complex scenarios. **Gauss-Seidel solvers** are robust and converge quickly for well-conditioned problems, but perform serial operations that do not scale well to large sets of constraints. **Jacobi solvers** perform parallel operations that can leverage GPU acceleration, but converge slower than the aforementioned techniques.

A naive implementation of Jacobi can outperform Gauss-Seidel for simulating numerous contact-rich interactions. However, we show that contact reduction can greatly accelerate Gauss-Seidel, achieving better performance than Jacobi.

4) Benchmark Problem: We pay special attention to the problem of accurately and efficiently simulating a nut-and-bolt assembly (**Fig. 2**). In the simulation community, this problem has emerged as a canonical challenge in contact-rich simulation. Moreover, in robotic assembly, tightening nuts onto bolts is critically important, as $\approx 40\%$ of all mechanical assembly operations involve screws and bolts [66].

The 3D finite element method (FEM) is the gold standard for accurate simulation of nut-and-bolt models, capturing deformation phenomena such as pretension ([39]). However, FEM simulation of a single nut-bolt pair may take $\sim$1-10 min to execute on CPU. As our aim is real-time (or faster) simulation to enable learning methods such as RL, we focus on rigid-body simulation prevalent in graphics and robotics.

In [105], rigid-body contact was simulated using semi-implicit integration, symbolic Gaussian elimination, analytical contact gradients, and an SVD solver. A dynamic scene was simulated of screwing a hex bolt into a threaded hole at $\frac{1}{460}$ real-time (6.0 h for 47 s). The proposed method was compared against Bullet (2012) [17], and a 7e4$\times$ speed-up was measured for a stable quasi-static version of the scene. In [24], rigid-body contact was simulated using a smoothed particle hydrodynamics solver that samples rigid surfaces with particles, models contacts as density deviations, and computes pressure-based contact forces implicitly. A quasi-static nut-and-bolt scene was simulated at $\frac{1}{20}$ real-time (81 s for 4 s). The proposed method was also compared against Bullet (2018), and a $9.5\times$ speed-up was measured for a stable scene.

In [22], rigid-body contact was simulated using an extension of incremental potential contact (IPC) [50] and continuous collision detection (CCD) for curved trajectories. A dynamic, frictionless nut-and-bolt scene was simulated at $\frac{1}{350}$ real-time (3.5 s for $\Delta t = 0.01$ s). The proposed method was compared against Bullet (2019), MuJoCo [94], Chrono [91], and Houdini RBD [85]. Instabilities were observed with Bullet and MuJoCo, and interpenetrations were observed with Chrono and Houdini. In [86], rigid-body contact was simulated using a configuration-space contact model. A single large M48 (48 mm diameter) nut-and-bolt was simulated at $4\times$ real-time; however, data and evaluations were limited. Informally, an industry group simulated a dynamic nut-and-bolt scene at $\approx\frac{1}{20}$ real-time with variational integrators [10, 44].

From the previous works, we define the established state-of-the-art for a general-purpose physics simulator that can simulate a nut-and-bolt assembly to be $\frac{1}{20}$ real-time for a single nut-bolt pair. Although fast, such speeds are not optimal for applications such as RL, which benefit from simulation of $\sim$1e2–1e3 contact-rich interactions in real-time. In this work, we demonstrate real-time simulation of 1000 nuts-and-bolts, as well as exceptional speeds on other challenging scenes.

B. Robotic Assembly Simulation

Here we review previous efforts to simulate robotic assembly, as well as efforts to use these simulators and/or real-world training for RL. For another recent review, see [88].

1) Simulation: Research efforts such as [105, 24, 22] have demonstrated advances in contact-rich simulation, often in comparison to physics engines used in robotics. However, the majority of robotics studies use PyBullet, MuJoCo, or Isaac Gym, as they provide robot importers, simulated sensors, parallel simulation, and learning-friendly APIs. Consequently, existing work in simulation for robotic assembly is largely limited to the performance of these simulators.

Efforts using MuJoCo or the robosuite extension [111] include [18, 21, 28, 36, 37, 47, 81, 87, 95, 96, 98, 107, 109, 110]. Simulated rigid-body assembly tasks are limited to peg-in-hole insertion of large pegs with round, triangular, square, and prismatic cross-sections, lap-joint mating, and one non-convex insertion [28]. Furthermore, clearances between rigid parts are typically 0.5-10 mm, substantially greater than real-world clearances. Efforts using PyBullet include [5, 56, 82, 106]. Simulated tasks are again limited to large peg-in-hole insertion and lap-joint mating, with clearances of $\sim$1 mm. A handful of research efforts have used other off-the-shelf simulators [7, 31, 46, 103, 110] and custom simulators [16, 57, 86]. Only [86] simulated a nut-and-bolt, which was exceptionally large (48 mm diameter) with inherently-larger clearances.

From the previous works, we conclude that there have been few successful efforts to simulate assembly tasks with realistic scales, realistic clearances (*e.g.*, $\sim$0.1 mm diametral clearance

for a $4\ mm$ peg), and complex geometries (*e.g.*, nuts-and-bolts, electrical connectors) within a robotics simulator. In this work, we build a module for PhysX and Isaac Gym that can successfully simulate all rigid components on NIST Task Board 1 [41] with accurate models and real-world clearances.

2) Reinforcement Learning: The studies in the previous subsection, as well as a small number of studies that trained RL for robotic assembly purely in the real world, can be categorized according to their choice of RL algorithm.

Several earlier works used model-based RL algorithms or proposed variants, which explicitly predict environment response during policy learning and/or execution. These efforts leveraged guided policy search [53, 92] and iterative LQG [54]. Such algorithms are sample-efficient, but difficult to apply to contact-rich tasks due to highly discontinuous and nonlinear dynamics and unknown material parameters [87].

Most recent works have used model-free, off-policy RL algorithms or variants, which do not predict environment response, and update an action-value function independently of the current policy (*e.g.*, using a replay buffer). These studies have applied Q-learning [34], deep-Q networks [109], deep deterministic policy gradients (DDPG) [5, 57, 55, 96], soft-actor critic [7], probabilistic embeddings [81], and hierarchical RL [31]. These algorithms are typically chosen for sample efficiency, but are often brittle and slow/unable to converge.

Several other studies use or develop off-policy RL algorithms that leverage human demonstrations, as well as motion planners and trajectory optimizers. These efforts have used residual learning from demonstration [18], guided DDPG [21], DDPG from demonstration (DDPGfD) [55, 56, 95], and inverse RL [103]. Notably, [55] used DDPGfD to achieve state-of-the-art performance in the real world for 3 insertion tasks from NIST Task Board 1. They used human demonstrations, real-world training, and human on-policy corrections, achieving a 99.8% success rate over 13k trials. Of course, these methods also require demonstration collection or effective planners/optimizers, and are typically limited to the performance of these structured priors.

Finally, a handful of research efforts have successfully used on-policy algorithms or variants. These studies have applied proximal policy optimization (PPO) [28, 86, 98], trust region policy optimization [46], asynchronous advantage actor-critic [82], and additional algorithms [37]. These algorithms are typically stable, easy-to-use, and achieve high return, but are highly sample-inefficient and require long wall-clock time.

We take inspiration from the performance of [55], but aim to achieve such performance in a fundamentally different way. We build a module for contact-rich simulation that can help enable roboticists to perform more complicated tasks (*e.g.*, nut-and-bolt assembly) with tight clearances; leverage performant and stable on-policy RL algorithms with high parallelization; avoid tedious human demonstrations and corrections; and mitigate the need for time-consuming (*e.g.*, $\sim$50 hours of data in [55]), costly, and dangerous real-world RL training.

III. Contact-Rich Simulation Methods

In this work, we first extend PhysX [74] to perform accurate and efficient contact-rich simulation. Specifically, we uniquely combine SDF collisions [60], contact reduction [71], and a Gauss-Seidel solver [59], allowing us to simulate interactions of highly-detailed models substantially faster than previous efforts. We describe critical methods and results below.

A. SDF Collisions

An SDF is a function $\phi(\boldsymbol{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ that maps a point $\boldsymbol{x}$ in Cartesian space to its Euclidean distance to a surface. The surface is implicitly defined by $\phi(\boldsymbol{x}) = 0$, and the sign indicates whether $\boldsymbol{x}$ is inside or outside the surface. The gradient $\nabla\phi(\boldsymbol{x})$ provides the normal at a point $\boldsymbol{x}$ on the surface. Collectively, the SDF value and gradient define a vector to push a colliding object out of the surface.

We generate the SDF for an object via sampling and compute gradients via finite-differencing. Specifically, given a triangle mesh representing the boundaries of the object, we generate an SDF for the mesh at initialization time and store it as a 3D texture, which enables GPU-accelerated texture fetching with trilinear filtering and extremely fast $\mathcal{O}(1)$ lookups. Since our shapes contain many small features, we typically use SDF resolutions of 256^3 or greater.

To generate an initial set of contacts, we use the method of [60], which generates one contact per triangle-mesh face. The contact position on each face is determined by performing iterative local minimization to find the closest point on the face to the opposing shape, using projected gradient descent with adaptive stepping. As an example, detailed M4 nut-and-bolt meshes generate $16k$ contacts; with the above method, these contacts can be generated in $< 1\ ms$, orders of magnitude faster than typical approaches for convex or mesh collision.

B. Contact Reduction and Solver Selection

To motivate our contact reduction methods, we begin with a brief discussion and first-order analysis of the tight coupling between contact generation and solver execution.

In a typical contact generation scheme, each contact only requires approximately 7 floats (point, normal vector, and distance) and 2 integers (rigid body indices), for a total of 36 bytes. However, the memory required to store constraints associated with these contacts is substantially greater; in our implementation, storing a contact and its constraints requires approximately 160 bytes. Thus, simulating an M4 nut-bolt pair with $16k$ contacts requires approx. $2.5\ MB$ per timestep.

For a Jacobi solver with $\Delta t = \frac{1}{60}s$, we require 8 substeps and 64 iterations for stable simulation. Thus, memory bandwidth requirements for $16k$ contacts are approximately $1.28\ GB$ per frame and $76.8\ GB$ per second. Using a state-of-the-art GPU, we can only simulate 20 nut-and-bolt assemblies in parallel (**Table V**). Unfortunately, although a Gauss-Seidel solver converges faster (*i.e.*, requires fewer substeps and iterations), it would be unreasonably slow for so many contacts due to its inherently serial nature. As reducing per-contact memory

can be challenging, contact reduction becomes a compelling strategy for reducing memory and increasing parallelization.

If we can reduce the number of contacts to 2% (*i.e.*, from $16k$ to 300), the simulation now requires approximately $48\ kB$ per timestep. The Jacobi solver now only requires $24\ MB$ per frame and $1.44\ GB$ per second. We can now hypothetically simulate a maximum of 1100 nut-and-bolt assemblies in real-time. Solving contact constraints is no longer a performance bottleneck, and we can achieve a level of parallelization suitable for training on-policy RL algorithms.

In addition, given the far fewer contacts, it is now feasible to use a Gauss-Seidel solver. With $\Delta t = \frac{1}{60}s$, we require only 1 substep and 16 iterations for stable simulation. Thus, we can now comfortably simulate > 1000 nut-and-bolt assemblies in real-time. We demonstrate exactly such performance later.

C. Implementation of Contact Reduction

To implement contact reduction, we use the concept of *contact patches*, which are a set of contacts that are proximal and share a normal. We generate contact patches in 3 phases (**Algorithm 1**). First, we **Generate** candidate contacts using SDF collisions for a batch of triangles (size 1024). Second, we **Assign** candidates to existing patches in the shape pair based on normal similarity. Third, for unassigned candidates, we **FindDeepest** (*i.e.*, find the one with deepest penetration), create a new patch, **BinReduce** (*i.e.*, assign remaining candidates to this patch based on normal similarity), and **AddPatch** to our list of patches. We repeat until no candidates remain. When performing **AddPatch**, we also check if a *patch* with a similar normal exists. We either add both patches or replace the existing patch, using a measure that maximizes patch surface area, prioritizes contacts with high penetration depth, and restricts the number of patches to N (where $128 \leq N \leq 256$).

The preceding contact reduction process is performed exclusively in GPU shared memory. Notably, contact reduction does not make contact generation slower; to the contrary, it makes generation faster, as the contact generation kernel does not need to write extensive amounts of data to global memory.

Applying the above procedure to the M4 nut-and-bolt interactions, we reduce the number of contacts from $16k$ to 300 (**Fig. S13**), allowing us to simulate 1024 assemblies in real-time on an NVIDIA A5000 GPU. Generating and reducing contacts takes 11 ms, and solving contact constraints takes an additional 3 ms. Further evaluations follow.

D. Performance Evaluations

We test our collision detection, contact generation, contact reduction, and solution pipeline on 8 contact-rich scenes. These scenes were designed to represent a broad range of challenging real-world scenarios, including complex geometries, robot-object interactions, tight clearances, 100s of interacting bodies, and multi-part mechanisms. The scenes are as follows:

- 1024 parallel $4\text{-}mm$ **peg-in-hole assemblies** from the NIST board with ISO-standard clearances ($0.104\ mm$).
- 1024 parallel **M16 nut-and-bolt assemblies** with ISO-standard clearances (**Fig 3**). For ease of contact profiling, the coefficient of friction is reduced to 0.05, allowing the nuts to rotate on the bolts under gravity.
- 1024 parallel VGA-style **D-subminiature (D-sub) connectors** from the NIST board (**Fig S14**). For ease of contact profiling, a 2e-6 m clearance is introduced, allowing the plug-and-socket to mate under gravity.
- 1024 parallel 2-stage **gear assemblies** from the NIST board (**Fig S14**). An external torque is applied to the intermediate gear to rotate the adjacent gears.
- 1024 **M16 nuts**, falling into a pile in one environment (**Fig. S15**).
- 1024 **bowls** (akin to [105]), falling into a pile in one environment. To enable larger timesteps while maintaining accuracy, a tiny 1e-6 m *negative* clearance is added.
- 1024 **toruses**, falling into a pile in one environment.
- 128 parallel **Franka robot + M16 nut-and-bolt assemblies**. A vibratory feeder mechanism feeds an aggregate of nuts into a channel. The robot grasps the nut from the channel and tightens it onto the bolt using an inverse kinematics (IK) controller (**Fig. 4**, **App. C2**).

Table VI describes the geometric representations used in each of the scenes, including SDF resolution and number of triangles. **Table I** provides statistics on contacts before and

Algorithm 1: Collision Detection: An overview of our collision detection pipeline. The input is two potentially contacting shapes (a, b). The output is a list of patches (*i.e.*, sets of contacts) for each shape pair.

foreach *shape pair* (a, b) **do**
  $patches = \emptyset$
  foreach *batch in triangles(a, b)* **do**
    $candidates \leftarrow$ GenerateContacts$(batch)$
    $patches \leftarrow$ Assign$(candidates, patches)$
    while *candidates* $\neq \emptyset$ **do**
      $best \leftarrow$ FindDeepest$(candidates)$
      $patch \leftarrow$ BinReduce$(candidates, best)$
      $patches \leftarrow$ AddPatch$(patch, patches)$
    end
  end
end

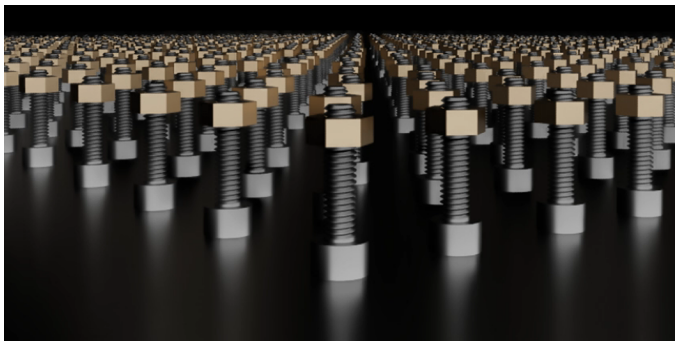

Fig. 3. Rendering of the **M16 nut-and-bolt assemblies** scene, consisting of 1024 parallel nut-and-bolt interactions executing in real-time.

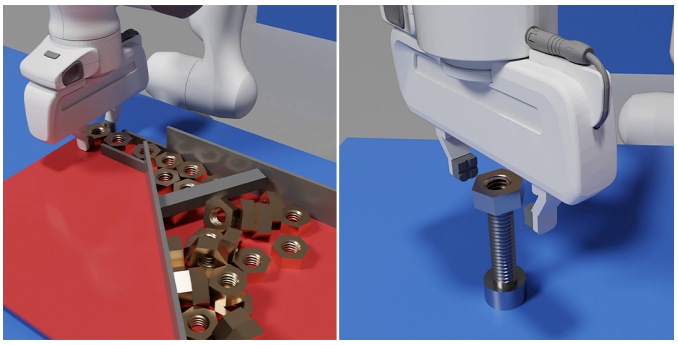

Fig. 4. Rendering of the **Franka robot + M16 nut-and-bolt assemblies** scene, consisting of 128 parallel Franka robots retrieving nuts from a vibratory feeder mechanism (left) and tightening them onto a bolt (right).

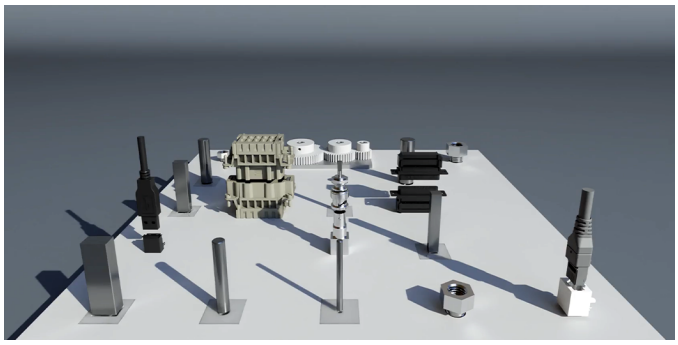

Fig. 5. Rendering of a simulated NIST Task Board 1, demonstrating the provided assets. We provide simulation and RL training environments for all rigid components of the board. Compare to the real board in **Fig. S16**.

after contact reduction, as well as timing for reduction and solution. Finally, **Table II** provides simulation performance statistics, including comparisons to real-time. We also qualitatively demonstrate an additional test scene: a **Franka robot + M16 nuts + flange assembly** scene (**App. C2**).

Although we defer to the tables for complete performance assessments, key observations include the following:

- Contact reduction can reduce contact counts by over 2 orders-of-magnitude compared to naive methods.
- Contact handling time (*i.e.*, pair finding, generation, reduction) is typically dominant compared to solution time.
- Parallelization achieves a 3-orders-of-magnitude speed-up over real-time single-threaded computation.

IV. Robot Learning Tools

We have thus far evaluated our physics simulation module over a diverse array of contact-rich scenes. However, the module is an extension of PhysX [74]. For convenient use in robot learning, we have integrated our module into Isaac Gym [63], which can use PhysX as its physics engine. To use our contact methods for arbitrary assets, the user simply has to include an $<sdf>$ tag in URDF descriptions (**Listing 1**).

For applications to robotic assembly, assets and scenes related to NIST Task Board 1 may be particularly useful. Thus, we provide 1) 60 assets from the NIST board, 2) 3 robotic assembly scenes for RL training in Isaac Gym, and 3) 7 real-world robot controllers in Isaac Gym to accelerate learning. Here we describe our assets, scenes, and controllers.

A. Assets

The NIST Task Board 1 consists of 38 unique parts. However, the CAD models publicly provided for these parts are not suitable for high-accuracy physics simulation. In particular, the models for the nuts, bolts, pegs, and gear assembly do not conform to real-world tolerances and clearances; in assembly, mating parts together with tight clearances is precisely the most significant challenge. Furthermore, the models for the electrical connectors were sourced from public repositories rather than manufacturers (*e.g.*, the D-sub plug and socket), were geometrically incompatible (*e.g.*, the RJ45 plug and socket, which interpenetrate), were incomplete (*e.g.*, the USB socket, which lacks mating features), and/or were designed using hand measurements (*e.g.*, the Waterproof plug and socket). Regardless of simulator accuracy, inaccurate geometries (esp. interpenetration) will lead to unstable or inaccurate dynamics.

We provide 60 high-quality, simulation-ready part models, each with an Onshape CAD model, one or more OBJ meshes, a URDF description, and estimated material properties (**Table VII** and **Table VIII**). These models include all the parts on the NIST Task Board 1, as well as dimensional variations. The assets for the nuts, bolts, pegs, and gearshafts conform to ISO 724, ISO 965, and ISO 286 standards and contain *loose* and *tight* configurations that correspond to the extremes of the tolerance band. The CAD models for the electrical connectors were sourced from manufacturer-provided models. Each part of each connector contains a *visual mesh*, directly exported from the CAD models, and a *collision mesh*, carefully redesigned to simplify external geometry while faithfully preserving mating features (*e.g.*, pins and holes).

B. Scenes

We provide 3 robotic assembly scenes for Isaac Gym that can be used for developing planning and control algorithms, collecting simulated sensor data for supervised learning, and training RL agents. Each scene contains a Franka robot and disassembled assemblies from NIST Task Board 1. All scenes have been tested with up to 128 simultaneous environments on an NVIDIA RTX 3090 GPU. The scenes are as follows:

- **FrankaNutBoltEnv**, which contains a Franka robot and nut-and-bolt assemblies of the user's choice (M4, M8, M12, M16, and/or M20). The nuts and bolts can be randomized in type and location across all environments. The default goal is to pick up a nut from a work surface and tighten it to the bottom of its corresponding bolt. Our own RL training results on this environment will be discussed in detail in the next section.
- **FrankaInsertionEnv**, which contains a Franka robot and insertion assemblies of the user's choice (round and/or rectangular pegs-and-holes; BNC, D-sub, and/or USB plugs-and-sockets) (**Fig. 6**). The assets can be randomized in type and location across all environments. The default

	Contact Stats (Before)			Contact Handling	Contact Stats (After)		Contact Solution
Scene	**Contacts**	**Per Pair (avg)**	**Per Pair (max)**	**Time**	**Per Pair (avg)**	**Patches**	**Time**
Peg-in-hole	5.89e5	576	576	2 ms	46	11	1 ms
Nut-and-bolt	1.73e7	16930	16930	11 ms	195	53	3 ms
D-sub connector	1.20e7	11746	11746	12.5 ms	175	36	1.5 ms
Gear assembly	7.26e7	14172	31568	39 ms	83	26	3 ms
Nuts	1.81e6	2516	8304	10 ms	99	46	2.1 ms
Bowls	5.23e5	731	1160	2 ms	66	18	4.3 ms
Toruses	4.00e5	185	864	4.6 ms	44	20	2.8 ms
Franka + nut-and-bolt	4.64e5	9285	10031	1.7 ms	147	40	2.7 ms

TABLE I. Contact statistics. *Contacts* is the total number of contacts in the scene. *Contact Handling* includes pair finding, contact generation, and contact reduction. *Contact Solution* includes contact constraint preparation, iterative constraint solution, and numerical integration.

		Timestepping		Simulation Stats	
Scene	**Substeps**	**Pos Iterations**	**Vel Iterations**	**Time**	**Real-time**
Peg-in-hole	1	4	1	3 ms	5689x
Nut-and-bolt	1	20	1	14 ms	1219x
D-sub connector	4	4	1	14 ms	305x
Gear assembly	4	4	1	42 ms	102x
Nuts	1	16	1	12 ms	1.39x
Bowls	2	50	1	6.3 ms	1.32x
Toruses	1	16	1	7.4 ms	2.25x
Franka + nut-and-bolt	4	16	1	4.4 ms	121x

TABLE II. Performance statistics. The baseline timestep (before substepping) is $\frac{1}{60}$ *s*. *Pos Iterations* and *Vel Iterations* denote the number of position and velocity iterations in the Gauss Seidel solver. *Time* denotes total simulation time per frame, which is the sum of *Contact Handling* and *Contact Solution* from **Table I**. *Real-time* denotes speed relative to the underlying dynamics of a single interaction; recall that most scenes contain numerous parallel environments.

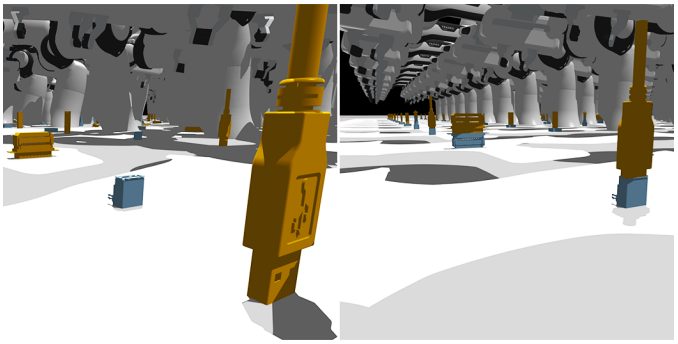

Fig. 6. Visualization of **FrankaInsertionEnv**. Each environment consists of a Franka robot and an insertion assembly from NIST Task Board 1. Left: The default initial state, where the positions of the parts are randomized on the work surface. Right: The default goal state, where all parts are inserted.

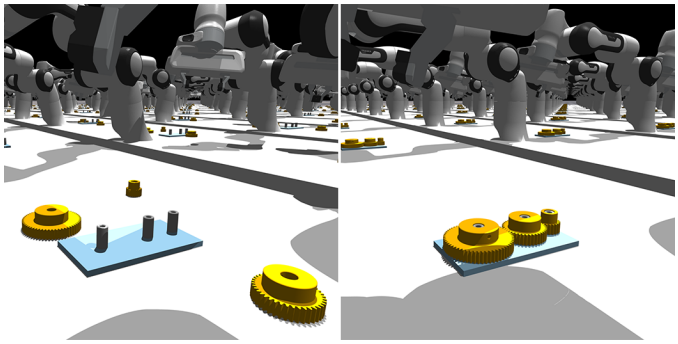

Fig. 7. Visualization of **FrankaGearsEnv**. Each environment consists of a Franka robot and the gear assembly from NIST Task Board 1. Left: The default initial state, where the positions of the gears are randomized on the work surface. Right: The default goal state, where all gears are aligned.

goal is to pick up a peg or plug and insert it into its corresponding hole or socket.

- **FrankaGearsEnv**, which contains a Franka robot and a 4-part gear assembly (**Fig. 7**). The assets can be randomized in location across all environments. The default goal is to pick up each gear, insert it onto its corresponding gear shaft, and align it with any other gears.

C. Controllers

Research efforts in reinforcement learning for robotic manipulation have traditionally used an action space consisting of low-level position, velocity, or torque commands. On the other hand, classical PD- or PID-style robot controllers have been used to solve contact-rich tasks in robotic assembly for several decades [68, 100]. In recent years, there has been substantial interest in using an RL action space consisting of targets to such controllers, with promising results in both sample efficiency and asymptotic performance [67, 76].

Akin to [111] in MuJoCo, we provide a suite of robot controllers based on those that researchers and engineers commonly use in the real world. The actions of the controllers are executed using an explicit integrator to avoid undesired damping. The controllers are as follows:

- **Joint-space inverse differential kinematics (IK) motion controller**, which converts task-space errors into joint-space errors and applies PD gains to generate joint torques. The IK controller can use either the geometric or analytic Jacobian [84] and generate torques with the Jacobian pseudoinverse, Jacobian transpose, damped least-squares (Levenberg-Marquardt), or adaptive SVD [11].

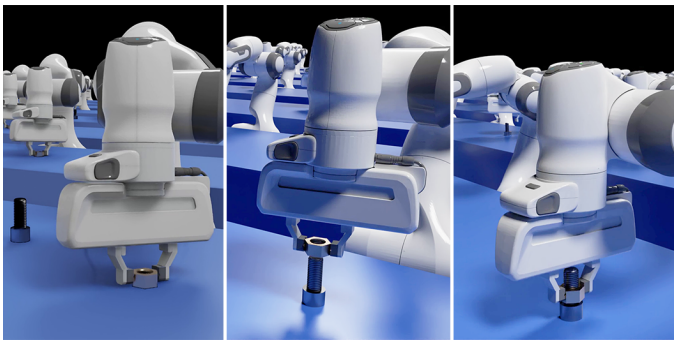

Fig. 8. Rendering of achieved goal states of our trained subpolicies for **FrankaNutBoltEnv**. Left: **Pick**. Middle: **Place**. Right: **Screw**.

- **Joint-space inverse dynamics (ID) controller**, which uses the joint-space inertia matrix and gravity compensation to generate joint torques, achieving desired spring-damper behavior in joint-space [78].
- **Task-space impedance controller**, which applies PD gains to task-space errors to generate joint torques. This controller is immediately available on the real-world Franka robot via the *libfranka* library [23].
- **Operational-space (OSC) motion controller**, which uses the task-space inertia matrix and gravity compensation to generate joint torques, achieving desired spring-damper behavior in task-space (akin to [101]).
- **Open-loop force controller**, which converts a task-space force target into joint torques.
- **Closed-loop P force controller**, which stacks an open-loop force controller with a closed-loop controller that applies P gains to task-space force errors.
- **Hybrid force-motion controller**, which stacks a task-space impedance or OSC motion controller with an open- or closed-loop force controller. Selection matrices can specify which axes use motion and/or force control.

Mathematical formulations are provided in **App. D4**.

V. Reinforcement Learning

The robotics community has demonstrated that RL can effectively solve simulated or real-world assembly tasks. However, these efforts are often limited to off-policy algorithms, require extensive training time or human demonstrations/corrections, and/or only address simple tasks. With our contact simulation methods, we use on-policy RL to solve the most contact-rich task on NIST Task Board 1: assembling a nut onto a bolt. Like many assembly tasks, such a procedure is long-horizon and challenging to learn end-to-end. We divide the task into 3 phases and learn an subpolicy for each:

- **Pick**: The robot grasps the nut with a parallel-jaw gripper from a random location on a work surface.
- **Place**: The robot transports the nut to the top of a bolt fixed to the surface.
- **Screw**: The robot brings the nut into contact with the bolt, engages the mating threads, and tightens the nut until it contacts the base of the bolt head.

RL is neither the only means to solve the 3 phases of this task, nor the most efficient: **Pick** and **Place** can be solved with classical grasping and motion controllers, and although challenging, **Screw** may be solved using a nut-driver and a compliance and/or suction mechanism [42]. We investigate this task as a proof-of-concept that our simulation methods can enable efficient policy learning for tasks of such complexity. Moreover, it is a common experience of simulation developers that model-free RL agents reveal and exploit any inaccuracies or instabilities in the simulator to maximize their reward; we view successfully training RL agents in contact-rich tasks as important qualitative evidence of simulator robustness.

We describe each subpolicy below; detailed evaluations will focus on **Screw**, the most contact-rich of the phases. We then address sequential execution and examine contact forces.

A. Shared Framework

The **Pick**, **Place**, and **Screw** subpolicies were all trained in Isaac Gym using our simulation methods and FrankaNutBoltEnv environment. The PPO implementation from [62] was used with a shared set of hyperparameters (**Table IX**). Typically, a batch of 3-4 policies were trained simultaneously on a single NVIDIA RTX 3090 GPU, with each policy using 128 parallel simulation environments. Each batch required a total of 1-1.5 hours for 1024 policy updates.

We define our action space as targets for our implemented controllers. Unless otherwise specified, the targets for the joint-space IK controller, joint-space ID controller, task-space impedance controller, and OSC motion controller were all 6-DOF transformations *relative* to the current state, with the rotation expressed as axis-angle. The targets for the open-loop and closed-loop force controller were 3-dimensional force vectors. The targets for the hybrid force-motion controller were the 9-dimensional union of the previous 2 action spaces.

We now discuss our randomization, observations, rewards, success criterion, and success rate for each subpolicy.

B. Subpolicy: Pick

At the start of each **Pick** episode, the 6-DOF Franka hand pose and 2-DOF nut pose (constrained by the work surface) were randomized over a large spatial range (**Table III**).

The observation space for **Pick** was the pose (position and quaternion) of the hand and nut, as well as the linear and angular velocity of the hand. In the real world, the pose and velocity of the hand can be determined to reasonable accuracy ($< 1\ cm$) through a forward kinematic model and proprioception of joint positions and velocities, whereas the pre-grasp pose of the nut (a known model, as typical in industrial settings) can be accurately estimated through pose estimation frameworks [43]. The action space for **Pick** consisted of joint-space IK controller targets with damped least-squares.

A dense reward was formulated as the distance between the fingertips and the nut. Initial experiments defined this distance as $||\mathbf{x}|| + \alpha||\mathbf{q}||$, where $\mathbf{x}$ and $\mathbf{q}$ are the translation and quaternion errors, and α is a scalar hyperparameter. However, this approach was sensitive to α. Inspired by [2], we reformulated

Pick		Place		Screw	
Parameter	**Range**	**Parameter**	**Range**	**Parameter**	**Range**
Hand X-axis	[-0.2, 0.2] m	Hand XY-axes	[-0.2, 0.2] m	Hand angle	[-90, 90] deg
Hand Y-axis	[-0.4, 0.0] m	Hand Z-axis	[0.5, 0.7] m	Fingertip X-axis	[-3, 3] mm
Hand Z-axis	[0.5, 0.7] m	Hand roll, pitch	[-17, 17] deg	Fingertip Y-axis	[-3, 3] mm
Hand roll, pitch	[-17, 17] deg	Hand yaw	[-57, 57] deg	Fingertip Z-axis	[0, 3] mm
Hand yaw	[-57, 57] deg	Nut-in-gripper XY-axes	[-2, 2] mm	Nut-in-gripper X-axis	[-3.5, 3.5] mm
Nut X-axis	[-0.1, 0.1] m	Nut-in-gripper Z-axis	[-5, 5] mm	Nut-in-gripper Z-axis	[-6.5, 1.0] mm
Nut Y-axis	[-0.4, 0.2] m	Nut-in-gripper yaw	[-180, 180] mm	Nut-in-gripper yaw	[-15, 15] deg
		Bolt XY-axes	[-10, 10] mm		

TABLE III. Ranges for initial randomization of pose parameters. Parameter values were uniformly sampled from the ranges.

the distance as $||\mathbf{k_n} - \mathbf{k_f}||$, where k_n and k_f are both tensors of 2-4 keypoints distributed along the nut central axis and end-effector approach axis, respectively. Intuitively, this method computes distance on a single manifold, obviating tuning. The collinearity of each keypoint set also allows equivariance to yaw rotation of the hand about the nut central axis.

After executing the **Pick** subpolicy for a prescribed (constant) number of timesteps, a manually-specified grasp-and-lift action was executed. Policy success was defined as whether the nut remained in the grasp after lifting. If successful, a success bonus was added to the episodic return.

With the above approach, the **Pick** policy was able to achieve a 100% success rate within the randomization bounds. Qualitatively, the agent learned to execute a fast straight-line path towards the nut, followed by a slow pose refinement. Due to the collinearity of the keypoints, the final pose distribution of the end-effector was highly multimodal in yaw.

C. Subpolicy: Place

At the start of each **Place** episode, the Franka hand and nut were reset to a known stable grasp pose. The nut-in-gripper position/rotation and the bolt position were randomized. The hand-and-nut were moved to a random pose using the joint-space IK controller (**Table III**). Training was then initiated.

The observation space for **Place** was identical to that for **Pick**, but also included the pose (position and quaternion) of the bolt. When grasped, the nut pose may be challenging to determine in the real world; however, recent research has demonstrated that visuotactile sensing with known object models can enable high-accuracy pose estimates [6].

The action space was identical to that for **Pick**. A dense reward was again formulated as a keypoint distance, now between the bolt and nut central axes. The keypoints were defined such that, when perfectly aligned, the base of the nut was located $1\ mm$ above the top of the bolt. Success was defined as when the average keypoint distance was $< 0.8\ mm$

With the above approach, the **Place** policy was able to achieve a 98.4% success rate within the randomization bounds. A common failure case during training was collision between the gripper and the bolt, dislodging the nut. Qualitatively, the robot learned trajectories that remained above the top plane of the bolt, with a slow pose refinement phase when close.

Although having negligible effect on steady-state error, an effective strategy for smoothing the **Place** trajectory was applying an *action gradient penalty* at each timestep. The penalty was equal to $\beta||\mathbf{a_t} - \mathbf{a_{t-1}}||$, where $\mathbf{a}$ is the 6-dimensional action vector and β is a hyperparameter (0.1).

D. Subpolicy: Screw

At the start of each **Screw** episode, the Franka hand and nut were reset to a stable grasp pose, randomized relative to the top of the bolt (**Table III**); these stable poses were generated using the **FrankaCalibrate** script described in **App. D3**. The nut-in-gripper position was also randomized as before.

Among the subpolicies, **Screw** was by far the most contact-rich, and as follows, challenging to train. The robot was required to bring the nut into contact with the bolt, engage the respective threads, generate gentle torques along the 7 arm joints to allow the high-inertia robot links to admit the rigid bolt constraint, and maintain appropriate posture of the gripper with respect to the nut during tightening. As a simplifying assumption, the joint limit of the end-effector was removed, allowing the Franka to avoid regrasping (akin to the Kinova Gen3). Nevertheless, training was replete with a diverse range of pathologies, including high-energy collision with the bolt shank, roll-pitch misalignment of the nut when first engaging the bolt threads, jamming of the nut during tightening, and precession of the gripper around the bolt during tightening, which induced slip between the gripper and nut.

To overcome the preceding issues, a systematic exploration of controllers/gains, observation/action spaces, and baseline rewards was executed. First, policies for 3 task-space controllers were evaluated over a wide range of gains, and the controller-gain configuration with the highest success rate was chosen (**Table X**). Then, 4 observation spaces were evaluated, and the space with the highest success rate was selected (**Table IV**). The procedure continued with 2 action spaces (**Table XI**) and 3 baseline rewards (**Table XII**). Success was defined as when the nut was less than 1 thread away from the base of the bolt.

To encourage stable robot posture, a dense reward was formulated that consisted of the sum of the keypoint distance [between nut and base of bolt] and [between end-effector and nut]. To prioritize both task completion and efficient training, early termination was applied on success and failure, and a maximum of 1024 gradient updates was allowed. Future work will investigate asymptotic performance with more updates.

Notably, collisions between the complex geometries of the nut and bolt remained stable during exploration by the RL agent. However, the majority of experimental groups failed due to the pathologies described earlier. The highest performing

agents consistently used an OSC motion controller with low proportional gains, an observation space consisting of pose and velocity of the gripper and nut, a 2-DOF action space (Z-translation and yaw), and a linear baseline reward. As expected, the relatively low number of epochs biased towards lower-dimensional observations and actions.

Using the above configuration, a final **Screw** policy was trained over 4096 gradient updates and achieved an 85.6% success rate over 1024 episodes.

E. Sequential Policy Execution

Although not our primary focus, a natural question arose on whether the subpolicies could be chained. Policy chaining can be challenging, as errors in each subpolicy can accumulate into poor overall performance; as a simple example, 3 perfectly-coupled subpolicies with 90% success rates can produce a combined policy with a 72.9% success rate.

In this work, we used a simple strategy to connect the learned **Pick**, **Place**, and **Screw** subpolicies end-to-end. Specifically, when training a given subpolicy, the initial states were randomized to span the distribution of the final states of the preceding trained subpolicy. For example, we defined the initial states of the **Screw** subpolicy (**Table III**) to span the maximum observed error of the **Place** subpolicy. For a small number of subpolicies, this strategy may be effective; however, the approach does not scale to long sequences, as Policy N must be trained and Sequence $0...N$ must be evaluated before training Policy $N+1$. To facilitate smoothness, an exponential moving average was applied on **Place** actions.

With this strategy, we achieved an end-to-end **Pick**, **Place**, and **Screw** success rate of 74.2%. More sophisticated techniques such as [15, 48] can be explored for improvements.

F. Contact Forces

Quantitatively and through numerous visual comparisons, our physics simulation module enabled accurate, efficient, and robust simulation of contact-rich interactions of assets with real-world geometries and material properties. Furthermore, our module was built on PhysX, which has been evaluated under challenging sim-to-real conditions (*e.g.*, [2, 80]).

Nevertheless, it is also important to consider the forces generated during such interactions. Although we leave a more detailed analysis to future work, we executed our **Screw** subpolicy and recorded joint torque norms, as well as contact force norms at the gripper fingers, nut, and bolt (**Fig. S17**). The joint torques are well within the range of lightweight collaborative robots (*e.g.*, UR3). Furthermore, the contact force norms at the fingertips were compared to analogous real-world forces from the Daily Interactive Manipulation dataset [33], in which human subjects tightened or loosened nuts with a wrench outfitted with a force-torque sensor [33] (**Fig. 9**).

Although the reward functions for the RL agents never involved contact forces, the robots learned policies that generated forces in the middle of human ranges; the much higher variance of human forces was likely due to more diverse strategies adopted by humans. Combined with our visual comparisons, these results do not guarantee sim-to-real policy transfer, but demonstrate that raw quantities computed by simulation are highly comparable to the real world.

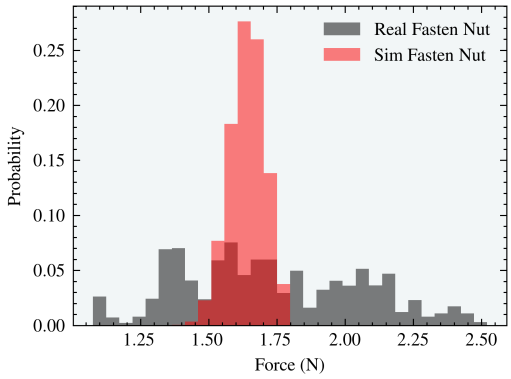

Fig. 9. Comparison of simulated contact forces during **Screw** subpolicy execution with analogous real-world contact forces from the Daily Interactive Manipulation (DIM) dataset. In this DIM task, humans tightened nuts using a wrench outfitted with a force-torque sensor. Maximum mean discrepancy (MMD) was 0.01269 (whereas values were $\sim$0.1-1.0 for unrelated tasks.)

VI. Discussion

We have presented **Factory**, a set of physics simulation methods and robot learning tools for contact-rich interactions in robotics. We provide a physics simulation module for PhysX and Isaac Gym that enables 100s to 1000s of contact-rich interactions to be simulated in real-time on a single GPU, as tested on a diverse array of scenes. As one example, 1000 nut-and-bolt interactions were simulated in real-time, whereas the prior state-of-the-art was a single nut-and-bolt at $\frac{1}{20}$ real-time.

We also provide 60 carefully-designed, ISO-standard or manufacturer-based assets from the NIST Assembly Task Board 1, suitable for high-accuracy simulation; 3 robotic assembly scenes in Isaac Gym where a robot can interact with these assets across a diverse range of assembly operations (fastener tightening, insertion, gear meshing); and 7 classical robot controllers that can achieve pose, force, or hybrid targets. We intend for our assets, scenes, and controllers to grow over time with contributions from the community.

Finally, we train proof-of-concept RL policies in Isaac Gym for the most contact-rich interaction on the board, nut-and-bolt assembly. We show that we can achieve stable simulator behavior, efficient training (1-1.5 hours to simultaneously train 4 policies on 1 GPU), high success rates, and realistic forces/torques. Although **Factory** was developed with robotic assembly as a motivating application, there are no limitations on using our methods for entirely different tasks within robotics, such as grasping of complex non-convex shapes in home environments, locomotion on uneven outdoor terrain, and non-prehensile manipulation of aggregates of objects.

VII. Limitations

We plan to address several limitations of this work. Within simulation, we plan to make 3 improvements to our SDF collision scheme: 1) the ability to robustly handle collisions

Observations	Success Rate	Env Steps to Success	Reward	Joint Torque (Nm)
Pose	0.7708	**2318**	-0.1019	**1.7319**
Pose, velocity	**0.7760**	3015	-0.0941	1.7330
Pose, velocity, force	0.5026	3849	-0.0784	**1.2186**
Pose, velocity, force, action	0.3307	3791	**-0.0521**	1.2257

TABLE IV. Comparison of observation spaces on performance of **Screw** task. *Success Rate* specifies the fraction of episodes that were successful. *Time to Success* specifies the mean number of timesteps required to achieve success. *Reward* specifies the mean reward during each episode. *Joint Torque* specifies the mean joint torque norm (Nm) during each episode. Each cell is computed from the average of 3 seeds. *Pose* was quickest; *Pose, velocity* exhibited highest success rate; and *Pose, velocity, force, action* achieved lowest mean reward, but did not consistently complete the task.

of thin-shell meshes (*e.g.*, thin-walled bottles and boxes), 2) improved handling of low-tessellation meshes, as currently, 1 contact is generated per-triangle, allowing penetration on large flat surfaces, 3) using sparse SDF representations to reduce the SDF memory footprint (**Table VI**). Furthermore, we are adding support for FEM-based simulation of stiff deformable features, such as the flexible tab on an RJ45 connector.

Within our assets, scenes, and controllers, we plan to add assets for additional industrial and home subassemblies (*e.g.*, USB-C, power plugs, key-in-lock), scenes for additional assembly tasks (*e.g.*, chain-and-sprocket assembly), and controllers found in industrial settings (*e.g.*, admittance). Within policy training, we plan to extend our policy for **FrankaNutBoltEnv** to learn regrasp behavior. In addition, we aim to develop a unified proof-of-concept policy for all insertion tasks within **FrankaInsertionEnv**, as well as a policy for meshing within **FrankaGearsEnv**, further evaluating training efficiency and simulator robustness. However, we encourage the broader RL community to test and develop state-of-the-art RL algorithms around these complex tasks.

A. Future Work

Upon making the aforementioned improvements, our future work will focus primarily on sim-to-real transfer. As described earlier, there has been compelling evidence sim-to-real is possible for industrial insertion tasks; we aim to demonstrate this for more complex bolt-tightening and gear-meshing tasks, as well as full $3D$ assembly operations in both industrial and home settings. For perception, we may train image-based policies using real-time ray-tracing and/or post-simulation path-tracing [8, 73] combined with domain randomization [93], However, we find distillation approaches to be particularly compelling. Specifically, we can

1) Train RL *teacher* policies, which take privileged information (*e.g.*, 6-DOF states) as input; and then use imitation learning to learn *student* policies, which take images as input and replicate the teacher's actions [13, 45], or
2) Train RL policies with the actor accessing images, but the critic accessing privileged information [4].

Adding noise on both low-dimensional and high-dimensional observations may be valuable. Furthermore, given that camera observations will be occluded during contact, we anticipate that integrating tactile sensing into our real-world system will be exceptionally critical for object-gripper pose estimation and slip detection.

Algorithmically, we are interested in obviating human demonstrations or corrections during policy learning. Nevertheless, contact-rich simulation from **Factory** is also suitable for simulation-based demonstration collection for use in imitation learning [16, 65] or DDPGfD-style policy learning.

VIII. Conclusion

We aim for **Factory** to establish the state-of-the-art in contact-rich simulation, as well as serve as an existence proof that highly efficient simulation and learning of contact-rich interactions is possible in robotics. Our experience has shown that high-quality assets and accurate, efficient simulation methods drastically reduce the inductive bias and algorithmic burden required to solve contact-rich tasks. We invite the community to establish benchmarks for solving the provided scenes, as well as extend and use **Factory** for their own contact-rich applications both within and outside of RL. We also hope that this work inspires researchers to execute contact-rich simulations of tasks beyond what we show in this paper to enable further solutions to complex problems.

Author Contributions

YN led the research project.

MM initially developed SDF collisions for FleX.

MM and YN conducted proof-of-concept demonstrations of SDF collisions for robotic assembly in FleX.

KS, ML, PR, and AM developed SDF collisions and contact reduction for PhysX.

PR, KS, AM, and YN developed evaluation and demonstration scenes for PhysX.

LW, YG, and GS integrated the PhysX developments into Isaac Gym.

YN, IA, and PR developed the assets, environments, and controllers for Isaac Gym.

YN, IA, and AH developed the RL policies within Isaac Gym.

DF, AH, MM, ML, GS, and AM advised the project.

Acknowledgments

We thank Joe Falco for assistance with the original NIST assets, John Ratcliff for generating the convex decomposition for the bolt (**Fig. S11**), Karl Van Wyk and Lucas Manuelli for helpful discussions on controllers, Viktor Makoviychuk for assistance with the *rl-games* [62] library, and the RSS reviewers for their insightful comments and questions.

References

[1] Afsoon Afzal, Deborah S. Katz, Claire Le Goues, and Christopher S. Timperley. A study on the challenges of using robotics simulators for testing. *arXiv:2004.07368 [cs]*, 2020.

[2] Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from GPU simulation to a remote real-world TriFinger. *arXiv:2108.09779 [cs]*, 2021.

[3] Sheldon Andrews and Kenny Erleben. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH Courses*, 2021.

[4] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *Int. J. Rob. Res.*, 2020.

[5] Aleksandra Anna Apolinarska, Matteo Pacher, Hui Li, Nicholas Cote, Rafael Pastrana, Fabio Gramazio, and Matthias Kohler. Robotic assembly of timber joints using reinforcement learning. *Autom. Constr.*, 2021.

[6] Maria Bauza, Eric Valls, Bryan Lim, Theo Sechopoulos, and Alberto Rodriguez. Tactile object pose estimation from the first touch with geometric contact rendering. In *Conference on Robot Learning (CoRL)*, 2020.

[7] Cristian C. Beltran-Hernandez, Damien Petit, Ixchel G. Ramirez-Alpizar, and Kensuke Harada. Variable compliance control for robotic peg-in-hole assembly: A deep-reinforcement-learning approach. *Appl. Sci.*, 2020.

[8] Blender Foundation. Blender: A 3d modelling and rendering package. http://www.blender.org, 2018.

[9] BNP Media. Assembly Magazine. https://www.assemblymag.com/, 2022. [Online; accessed 1-January-2022].

[10] Kenneth Bodin. https://twitter.com/kennethbodin/status/1252973291217850368, 2021. [Online; accessed 1-January-2022].

[11] Samuel R. Buss. Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. 2009.

[12] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[13] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning (CoRL)*, 2021.

[14] HeeSun Choi, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory Hager, David Han, Frank Hearl, Jessica Hodgins, Abhinandan Jain, Frederick Leve, Chen Li, Franziska Meier, Dan Negrut, Ludovic Righetti, Alberto Rodriguez, Jie Tan, and Jeff Trinkle. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proc. Natl. Acad. Sci. USA*, 2021.

[15] Alexander Clegg, Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Learning to dress: Synthesizing human dressing motion via deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 2018.

[16] Henry M. Clever, Ankur Handa, Hammad Mazhar, Kevin Parker, Omer Shapira, Qian Wan, Yashraj Narang, Iretiayo Akinola, Maya Cakmak, and Dieter Fox. Assistive Tele-op: Leveraging transformers to collect robotic task demonstrations. *arXiv:2112.05129 [cs]*, 2021.

[17] Erwin Coumans. Bullet real-time physics simulation. https://pybullet.org/wordpress/, 2022. [Online; accessed 1-January-2022].

[18] Todor Davchev, Kevin Sebastian Luck, Michael Burke, Franziska Meier, Stefan Schaal, and Subramanian Ramamoorthy. Residual learning from demonstration: Adapting DMPs for contact-rich manipulation. *arXiv:2008.07682 [cs]*, 2021.

[19] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ManipulaTHOR: A framework for visual object manipulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[20] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. ACRONYM: A large-scale grasp dataset based on simulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[21] Yongxiang Fan, Jieliang Luo, and Masayoshi Tomizuka. A learning framework for high precision industrial assembly. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[22] Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M Kaufman, and Daniele Panozzo. Intersection-free rigid body dynamics. *ACM Trans. Graph.*, 2021.

[23] Franka Emika. libfranka examples. 2017.

[24] Christoph Gissler, Andreas Peer, Stefan Band, Jan Bender, and Matthias Teschner. Interlinked SPH pressure solvers for strong fluid-rigid coupling. *ACM Trans. Graph.*, 2019.

[25] Loeiz Glondu, Sara C. Schvartzman, Maud Marchal, Georges Dumont, and Miguel A. Otaduy. Efficient collision detection for brittle fracture. In *Proc. of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2012.

[26] Huy Ha and Shuran Song. FlingBot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning (CoRL)*, 2021.

[27] Kris Hauser. Robust contact generation for robot simulation with unstructured meshes. In *Robotics Research*. Springer International Publishing, 2016.

[28] Marius Hebecker, Jens Lambrecht, and Markus Schmitz. Towards real-world force-sensitive robotic assembly through deep reinforcement learning in simulations. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2021.

[29] Neville Hogan and Stephen Buerger. Impedance and Interaction Control. In *Robotics and Automation Handbook*. CRC Press, 2004.

[30] Peter C. Horak and Jeff C. Trinkle. On the similarities and differences among contact models in robot simulation. *IEEE Robot. Autom. Lett.*, 2019.

[31] Zhimin Hou, Jiajun Fei, Yuelin Deng, and Jing Xu. Data-efficient hierarchical reinforcement learning for robotic assembly control applications. *IEEE Trans. Ind. Electron.*, 2021.

[32] Isabella Huang, Yashraj Narang, Clemens Eppner, Balakumar Sundaralingam, Miles Macklin, Tucker Hermans, and Dieter Fox. DefGraspSim: Simulation-based grasping of 3D deformable objects. In *Robotics: Science and Systems (RSS) Workshop on Deformable Object Simulation in Robotics (DO-Sim)*, 2021.

[33] Yongqiang Huang and Yu Sun. A dataset of daily interactive manipulation. *The International Journal of Robotics Research*, 2019.

[34] Tadanobu Inoue, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana. Deep reinforcement learning for high precision assembly tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[35] Zhenzhong Jia, Ankit Bhatia, Reuben M. Aronson, David Bourne, and Matthew T. Mason. A survey of automated threaded fastening. *IEEE Trans. Automat. Sci. Eng.*, 2019.

[36] Shiyu Jin, Diego Romeres, Arvind Ragunathan, Devesh K. Jha, and Masayoshi Tomizuka. Trajectory optimization for manipulation of deformable objects: Assembly of belt drive units. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[37] Shahbaz Abdul Khader, Hang Yin, Pietro Falco, and Danica Kragic. Stability-guaranteed reinforcement learning for contact-rich manipulation. *IEEE Robot. Autom. Lett.*, 2021.

[38] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE J. Robot. Automat.*, 1987.

[39] Jeong Kim, Joo-Cheol Yoon, and Beom-Soo Kang. Finite element analysis and modeling of structure with bolted joints. *Applied Mathematical Modelling*, 2007.

[40] Myungsin Kim, Jaemin Yoon, Dongwon Son, and Dongjun Lee. Data-driven contact clustering for robot simulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[41] Kenneth Kimble, Karl Van Wyk, Joe Falco, Elena Messina, Yu Sun, Mizuho Shibata, Wataru Uemura, and Yasuyoshi Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE Robot. Autom. Lett.*, 2020.

[42] KUKA. Fast & reliable micro screw fastening. https://www.youtube.com/watch?v=BOY3oZ8SkZU, 2016. [Online; accessed 1-January-2022].

[43] Y. Labbe, J. Carpentier, M. Aubry, and J. Sivic. CosyPose: Consistent multi-view multi-object 6D pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[44] Claude Lacoursiere. *Ghosts and machines: regularized variational methods for interactive simulations of multibodies with dry frictional contacts*. PhD thesis, Umeå University, 2007.

[45] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 2020.

[46] Michelle A. Lee, Yuke Zhu, Peter Zachares, Matthew Tan, Krishnan Srinivasan, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Trans. Robot.*, 2020.

[47] Youngwoon Lee, Edward S. Hu, and Joseph J. Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[48] Youngwoon Lee, Joseph J Lim, Anima Anandkumar, and Yuke Zhu. Adversarial skill chaining for long-horizon robot manipulation via terminal state regularization. In *Conference on Robot Learning (CoRL)*, 2021.

[49] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, C. Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. iGibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv:2108.03272 [cs]*, 2021.

[50] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 2020.

[51] Wenzhao Lian, Tim Kelch, Dirk Holz, Adam Norton, and Stefan Schaal. Benchmarking off-the-shelf solutions to robotic assembly tasks. *arXiv:2103.05140 [cs]*, 2021.

[52] C. Karen Liu and Dan Negrut. The role of physics-based simulators in robotics. *Annu. Rev. Control Robot. Auton. Syst.*, 2021.

[53] Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, and Alice M. Agogino. Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects. In *IEEE/RSJ International Conference on Intelligent Robots and*

Systems (IROS), 2018.

[54] Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M. Agogino, Aviv Tamar, and Pieter Abbeel. Reinforcement learning on variable impedance controller for high-precision robotic assembly. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[55] Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Wenzhao Lian, Chang Su, Mel Vecerik, Ning Ye, Stefan Schaal, and Jon Scholz. Robust multi-modal policies for industrial assembly via reinforcement learning and demonstrations: A large-scale study. *arXiv:2103.11512 [cs]*, 2021.

[56] Jieliang Luo and Hui Li. Dynamic experience replay. In *Conference on Robot Learning (CoRL)*, 2019.

[57] Jieliang Luo and Hui Li. A learning approach to robot-agnostic force-guided high precision assembly. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[58] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[59] Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. Small steps in physics simulation. 2019.

[60] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. Local optimization for robust signed distance field collision. *Proc. ACM Comput. Graph. Interact. Tech.*, 2020.

[61] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Sci. Robot.*, 2019.

[62] Denys Makoviichuk and Viktor Makoviychuk. rl-games, 2021. URL https://github.com/Denys88/rl_games/.

[63] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High performance GPU-based physics simulation for robot learning. In *Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

[64] Khaled Mamou. Volumetric hierarchical approximate convex decomposition. In *Game Engine Gems 3*. A.K. Peters / CRC Press, 2016.

[65] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. RoboTurk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning (CoRL)*, 2018.

[66] Louis A. Martin-Vega, Harold K. Brown, Wade H. Shaw, and Thomas J. Sanders. Industrial perspective on research needs and opportunities in manufacturing assembly. *J. Manuf. Syst.*, 1995.

[67] Roberto Martín-Martín, Michelle A. Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[68] Matthew T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[69] Carolyn Matl, Yashraj Narang, Ruzena Bajcsy, Fabio Ramos, and Dieter Fox. Inferring the material properties of granular media for robotic tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[70] Carolyn Matl, Yashraj Narang, Dieter Fox, Ruzena Bajcsy, and Fabio Ramos. STReSSD: Sim-to-real from sound for stochastic dynamics. In *Conference on Robot Learning (CoRL)*, 2020.

[71] Ádám Moravánszky and Pierre Terdiman. Fast contact reduction for dynamics simulation. In Andrew Kirmse, editor, *Game Programming Gems 4*, pages 253–263. Charles River Media, 2004.

[72] Yashraj Narang, Balakumar Sundaralingam, Miles Macklin, Arsalan Mousavian, and Dieter Fox. Sim-to-real for robotic tactile sensing via physics-based simulation and learned latent projections. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[73] NVIDIA. NVIDIA Omniverse Platform. https://developer.nvidia.com/nvidia-omniverse-platform, 2022. [Online; accessed 1-January-2022].

[74] NVIDIA. NVIDIA PhysX SDK. https://github.com/NVIDIAGameWorks/PhysX, 2022. [Online; accessed 1-January-2022].

[75] M.A. Otaduy and M.C. Lin. A modular haptic rendering algorithm for stable and transparent 6-DOF manipulation. *IEEE Trans. Robot.*, 2006.

[76] Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using DeepRL: does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2017.

[77] Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.*, 2017.

[78] Robotic Systems Lab, ETH Zurich. Robot dynamics lecture notes. 2018.

[79] Alessio Rocchi, Barrett Ames, Zhi Li, and Kris Hauser. Stable simulation of underactuated compliant hands. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[80] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2021.

[81] Gerrit Schoettler, Ashvin Nair, Juan Aparicio Ojea, Sergey Levine, and Eugen Solowjow. Meta-reinforcement learning for robotic industrial insertion tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[82] Lin Shao, Toki Migimatsu, and Jeannette Bohg. Learning to scaffold the development of robotic manipulation skills. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[83] Zilin Si and Wenzhen Yuan. Taxim: An example-based simulation model for GelSight tactile sensors. *arXiv:2109.04027 [cs]*, 2021.

[84] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics*. Springer, 2009.

[85] SideFX. Houdini. https://www.sidefx.com/products/houdini/, 2022. [Online; accessed 1-January-2022].

[86] Dongwon Son, Hyunsoo Yang, and Dongjun Lee. Sim-to-real transfer of bolting tasks with tight tolerance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[87] Oren Spector and Miriam Zacksenhouse. Deep reinforcement learning for contact-rich skills using compliant movement primitives. *arXiv:2008.13223 [cs]*, 2020.

[88] Liliana Stan, Adrian Florin Nicolescu, and Cristina Pupăză. Reinforcement learning for assembly robots: A review. *Proc. Manuf. Syst.*, 2020.

[89] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham. Can robots assemble an IKEA chair? *Sci. Robot.*, 2018.

[90] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[91] Alessandro Tasora, Radu Serban, Hammad Mazhar, Arman Pazouki, Daniel Melanz, Jonathan Fleischmann, Michael Taylor, Hiroyuki Sugiyama, and Dan Negrut. Chrono: An open source multi-physics dynamics engine. In *High Performance Computing in Science and Engineering*. Springer International Publishing, 2016.

[92] Garrett Thomas, Melissa Chien, Aviv Tamar, Juan Aparicio Ojea, and Pieter Abbeel. Learning robotic assembly from CAD. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[93] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[94] 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*, 2012.

[95] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv:1707.08817 [cs]*, 2018.

[96] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothorl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[97] Felix von Drigalski, Christian Schlette, Martin Rudorfer, Nikolaus Correll, Joshua C. Triyonoputro, Weiwei Wan, Tokuo Tsuji, and Tetsuyou Watanabe. Robots assembling machines: learning from the World Robot Summit 2018 Assembly Challenge. *Adv. Robot.*, 2019.

[98] Nghia Vuong, Hung Pham, and Quang-Cuong Pham. Learning sequences of manipulation primitives for robotic assembly. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[99] Shaoxiong Wang, Mike Lambeta, Po-Wei Chou, and Roberto Calandra. TACTO: A fast, flexible and open-source simulator for high-resolution vision-based tactile sensors. *arXiv:2012.08456 [cs, stat]*, 2020.

[100] Daniel E. Whitney. *Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development*. Oxford University Press, 2004.

[101] Josiah Wong, Viktor Makoviychuk, Anima Anandkumar, and Yuke Zhu. OSCAR: Data-driven operational space control for adaptive and robust robot manipulation. *arXiv:2110.00704 [cs]*, 2021.

[102] Bohan Wu, Iretiayo Akinola, Jacob Varley, and Peter Allen. MAT: Multi-fingered adaptive tactile grasping via deep reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

[103] Zheng Wu, Wenzhao Lian, Vaibhav Unhelkar, Masayoshi Tomizuka, and Stefan Schaal. Learning dense rewards for contact-rich manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[104] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[105] Hongyi Xu, Yili Zhao, and Jernej Barbič. Implicit multibody penalty-based distributed contact. *IEEE Trans. Visual. Comput. Graphics*, 2014.

[106] Mingxin Yu, Lin Shao, Zhehuan Chen, Tianhao Wu, Qingnan Fan, Kaichun Mo, and Hao Dong. RoboAssembly: Learning generalizable furniture assembly policy in a novel multi-robot contact-rich sim-

Fig. S10. Geometric representations, demonstrated on an M4 bolt. Left to right: Convex hull. Convex decomposition generated via V-HACD [64] (951 shapes). Triangle mesh representation generated in Onshape ($47k$ triangles). SDF representation, visualized as a mesh of the isosurface.

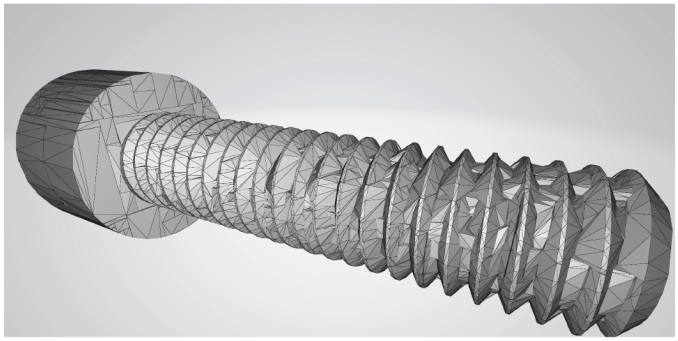

Fig. S11. Automatic convex decomposition for an M4 bolt. The decomposition consists of 951 convex shapes. Spatial artifacts are apparent.

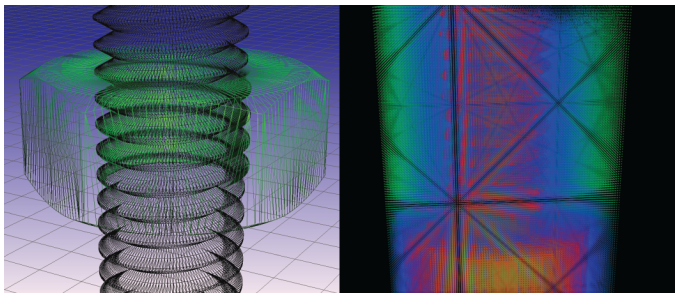

Fig. S12. Left: High-quality triangular mesh representation of an M4 nut and bolt, consisting of $27k$ tris (nut) and $47k$ tris (bolt). Right: SDF representation of the bolt, stored in a voxel grid of dimensions $144\times256\times144$ and visualized as a point cloud. For easier visualization, colors are interpolated according to $\phi(\boldsymbol{x})^{\frac{1}{4}}$. Streaks are artifacts of the viewing angle.

ulation environment. *arXiv:2112.10143 [cs]*, 2021.

[107] Peter A. Zachares, Michelle A. Lee, Wenzhao Lian, and Jeannette Bohg. Interpreting contact interactions to overcome failure in robot assembly tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[108] Mabel M. Zhang. Necessity for more realistic contact simulation. In *Robotics: Science and Systems (RSS) Workshop on Visuotactile Sensors for Robust Manipulation*, 2020.

[109] Xiang Zhang, Shiyu Jin, Changhao Wang, Xinghao Zhu, and Masayoshi Tomizuka. Learning insertion primitives with discrete-continuous hybrid action space for robotic assembly tasks. *arXiv:2110.12618 [cs]*, 2021.

[110] Jialiang Zhao, Daniel Troniak, and Oliver Kroemer. Towards robotic assembly by predicting robust, precise and task-oriented grasps. *arXiv:2011.02462 [cs]*, 2020.

[111] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv:2009.12293 [cs]*, 2020.

APPENDIX

A. Introduction

No supplementary information.

B. Related Works

No supplementary information.

C. Contact-Rich Simulation Methods

1) Initial Nut-and-Bolt Experiments: We performed initial experiments with various geometric representations of nuts and bolts before proceeding with SDFs. **Fig. S10** shows 4 different geometric representations of an M4 bolt mesh.

The convex representation is clearly insufficient to simulate contact-rich interactions. The convex decomposition, generated with voxel-based V-HACD using Omniverse [73], consists of 951 convex hulls and appears to faithfully represent the exact bolt geometry. However, **Fig. S11** shows a closer view; artifacts are prevalent. These artifacts cause inaccuracy and instability when simulating contact with a mating M4 nut.

The trimesh representation, consisting of $47k$ triangles, is highly faithful to the exact geometry. **Fig. S12** shows the M4 bolt and nut meshes. Initial experiments were conducted with trimesh-trimesh collisions using boundary-layer expanded meshes [27]. However, stable behavior was only possible with extremely small timesteps ($\Delta t = 1e\text{-}3$ s with 10 substeps), and the resulting simulations executed at $\frac{1}{80}$ of real-time.

Only SDF representations enabled accurate, efficient, and robust simulation. **Fig S12** shows a point-cloud visualization of the SDF for the M4 bolt mesh. However, generating SDFs is an expensive process proportional to the number of samples, and storage requires significant memory. The M4 bolt model consists of 5.3 million samples, each a floating-point distance.

2) Additional Scene Descriptions: **Franka robot + M16 nut-and-bolt assemblies**. A vibratory feeder is an ubiquitous mechanism for conveying and isolating mechanical components. The feeder consists of a tray or bowl that vibrates an aggregate of parts at high-frequency and small-amplitude. Under gravity, the parts gradually move towards a singulation mechanism, which isolates them for inspection or downstream handling. We demonstrate a tray-style inclined feeder that vibrates an aggregate of nuts at 60 Hz. The nuts towards a channel that only allows one nut to pass at a time. A Franka robot with a hand-scripted controller then grasps a nut from

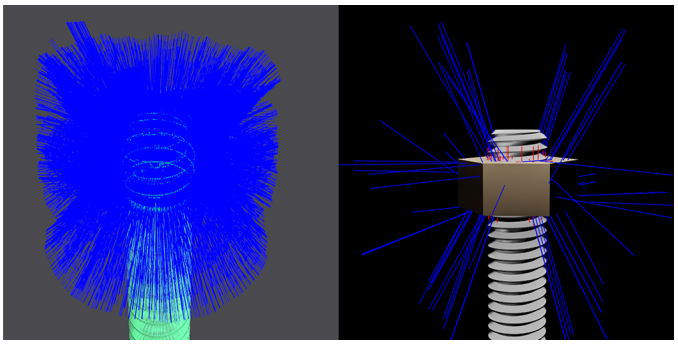

Fig. S13. Contacts generated between a nut and bolt mesh before and after applying our contact reduction scheme. Left: $16k$ contacts generated on bolt before reduction. Right: 300 contacts remaining after reduction.

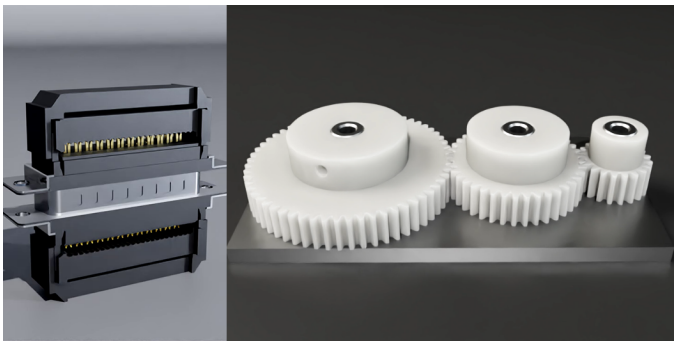

Fig. S14. Renderings of D-sub connector and gear assembly assets used in simulation evaluations and provided in this work.

the channel opening and tightens it onto a bolt (**Fig. 4**).

Franka robot + M16 nuts + flange assembly. This scene is similar to the above, except that the Franka grasps and tightens multiple nuts in sequence to clamp together 2 halves of a flange (**Fig. S15**). This scene is only qualitatively evaluated.

3) Rendering: For visualization purposes, we render several figures using Omniverse. The details are as follows:

- The output of the simulation in **Fig. 1** was path-traced.
- The simulation in **Fig. 3** was rendered in real-time.
- The scene in **Fig. 5** was path-traced.
- The scenes in **Fig. 8** were path-traced.
- The scenes in **Fig. S14** were path-traced.

Fig. S15. Renderings of 2 scenes. Left: **M16 nuts** scene. Right: **Franka robot + M16 nuts + flange assembly scene.**

- The scene in **Fig. S15** was path-traced.

For simulation-related figures not listed above, the images were captured directly from PhysX, Isaac Gym, or Onshape.

D. Robot Learning Tools

1) Assets: **Table VII** and **Table VIII** provide details on all assets provided in this work. We intend for this to be a growing database, with contributions from the community.

2) Using Contact Simulation Methods: To use the contact simulation methods from this work in Isaac Gym, the user simply needs to add an $<sdf>$ tag to the $<collision>$ block of the object that should use SDF collisions (**Listing 1**). PhysX will then interpret the $<sdf>$ tag as follows:

Consider 2 colliding objects, Object A and Object B.

- If A and B both have an $<sdf>$ tag, SDF-mesh collision will be applied. The object with the larger number of features (*i.e.*, triangles) will be represented as an SDF, and the trimesh of the other object will be queried against the SDF to check for collisions and generate contacts.
- If A has an $<sdf>$ tag and B does not, convex-mesh collision will be applied. Object A will be represented as a trimesh, and object B will be represented as a convex.
- If neither A nor B has an $<sdf>$ tag, a default convex-convex collision will be applied.

```
<?xml version="1.0"?>
<robot name="nut_m16">
    <link name="nut_m16">
        <visual>
            <geometry>
                <mesh filename="nut_m16_tight.obj"/>
            </geometry>
        </visual>
        <collision>
            <geometry>
                <mesh filename="nut_m16_tight.obj"/>
            </geometry>
            <sdf resolution="256"/>
        </collision>
    </link>
</robot>
```

Listing 1. Example URDF file for a nut asset with SDF collision enabled. The *resolution* field specifies the desired number of voxels along the longest dimension of the object. We have typically used 256 or 512.

3) Helpful Additions: For all assembly assets provided (*e.g.*, nut and bolt), the zero-positions of the parts correspond to the configuration in which they are fully assembled; thus, specifying a goal state for RL is extremely simple for the user.

Furthermore, we provide a helper script called **FrankaCalibrate**. When training RL policies, it is common to randomize the joint state of the robot at the beginning of each episode in order to facilitate robustness. Simultaneously, if a robot is learning a complex multi-step assembly procedure, it can be helpful to learn short-horizon sub-policies that begin when an object is already grasped. Nevertheless, in practice, it is challenging to randomize the state of a robot *and* a grasped object *while maintaining grasp stability*. Using **FrankaCalibrate**, the Franka can grasp an object, go to randomized poses, and generate a large set of stable robot-and-object poses that can be imported when subsequently training policies.

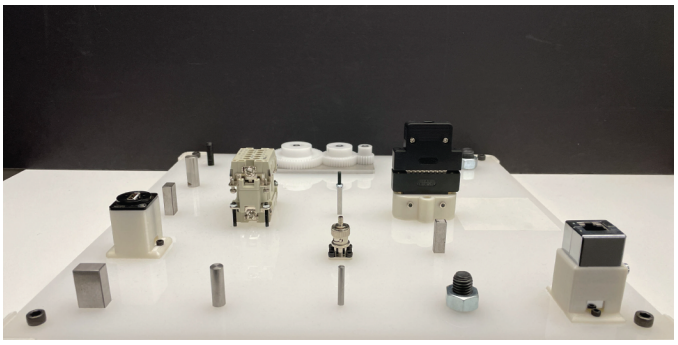

Fig. S16. Image of a real NIST Task Board 1.

4) Controllers: The controllers implemented in this work are based on the mathematical formulations articulated in [11, 29, 58, 78, 84]. Our specific control laws are as follows:

- **Joint-space inverse differential kinematics (IK) motion controller**

$$\boldsymbol{\tau} = \mathbf{k_p}(\mathbf{q_t} - \mathbf{q_c}) - \mathbf{k_d}(\dot{\mathbf{q}}_\mathbf{c}) \tag{1}$$

  where $\boldsymbol{\tau}$ is the joint torque vector ($\mathbb{R}^7$), $\mathbf{k_p}$ and $\mathbf{k_d}$ are diagonal matrices consisting of joint-space proportional gains ($\mathbb{R}^{7\times7}$) and joint-space derivative gains ($\mathbb{R}^{7\times7}$), $\mathbf{q}$ and $\dot{\mathbf{q}}$ are the joint position vector ($\mathbb{R}^7$) and joint velocity vector ($\mathbb{R}^7$), and subscripts $\mathbf{t}$ and $\mathbf{c}$ denote *target* and *current*. The quantity $\mathbf{q_t} - \mathbf{q_c}$ is computed by inputting a task-space error into an IK algorithm. For IK, we have implemented Jacobian pseudoinverse, Jacobian transpose, damped least-squares (Levenberg-Marquardt), and adaptive singular value decomposition (SVD) [11].

- **Joint-space inverse dynamics controller**

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{k_p}(\mathbf{q_t} - \mathbf{q_c}) - \mathbf{k_d}(\dot{\mathbf{q}}_\mathbf{c})) + \mathbf{b} + \mathbf{g} \tag{2}$$

  where M is the joint-space inertia matrix ($\mathbb{R}^{7\times7}$), $\mathbf{b}$ is the joint-space Coriolis and centrifugal torque vector ($\mathbb{R}^7$), and $\mathbf{g}$ is the joint-space gravitational torque vector (*i.e.*, a gravity compensation term) ($\mathbb{R}^7$). Vector $\mathbf{b}$ is currently ignored due to small joint velocities during assembly.

- **Task-space impedance controller**

$$\boldsymbol{\tau} = \mathbf{J^T}(\mathbf{k_p}(\mathbf{x_t} \ominus \mathbf{x_c}) - \mathbf{k_d}(\dot{\mathbf{x}}_\mathbf{c})) \tag{3}$$

  where $\mathbf{J}$ is the geometric or analytic Jacobian ($\mathbb{R}^{6\times7}$), $\mathbf{k_p}$ and $\mathbf{k_d}$ are now diagonal matrices consisting of task-space proportional gains ($\mathbb{R}^{6\times6}$), and $\mathbf{x}$ and $\dot{\mathbf{x}}$ are the task-space position vector ($\mathbb{R}^6$) and task-space velocity vector ($\mathbb{R}^6$). The analytic Jacobian can be computed from the geometric Jacobian as described in [78]. When using the geometric Jacobian, the orientation components of $\dot{\mathbf{x}}$ are angular velocity, and $\ominus$ computes the rotational transformation from current to target. When using the analytic Jacobian, the orientation components of $\dot{\mathbf{x}}$ are the derivatives of axis-angle, and $\ominus$ simply computes the difference between target and current.

- **Operational-space motion controller**

$$\boldsymbol{\tau} = \mathbf{J^T}(\mathbf{M}(\mathbf{k_p}(\mathbf{x_t} - \mathbf{x_c}) - \mathbf{k_d}(\dot{\mathbf{x}}_\mathbf{c})) + \mathbf{b} + \mathbf{g}) \tag{4}$$

  where M is now the task-space inertia matrix ($\mathbb{R}^{6\times6}$), $\mathbf{b}$ is now the task-space Coriolis and centrifugal torque vector ($\mathbb{R}^6$), and $\mathbf{g}$ is now the task-space gravitational torque vector ($\mathbb{R}^6$). The task-space inertia matrix can be computed from the joint-space inertia matrix [38]. As before, vector $\mathbf{b}$ is currently ignored due to small velocities during assembly operations.

- **Open-loop force controller**

$$\boldsymbol{\tau} = \mathbf{J^T}\mathbf{F_t} \tag{5}$$

  where $\mathbf{F_t}$ is the target wrench ($\mathbb{R}^6$).

- **Closed-loop P force controller**

$$\boldsymbol{\tau} = \mathbf{J^T}(\mathbf{F_t} + \mathbf{k_f}(\mathbf{F_t} - \mathbf{F_c})) \tag{6}$$

  where $\mathbf{k_f}$ is a diagonal matrix consisting of task-space proportional gains ($\mathbb{R}^{6\times6}$).

- **Hybrid force-motion controller**

$$\begin{aligned}\boldsymbol{\tau} = \mathbf{J^T}(\mathbf{S_m}[\mathbf{M}(\mathbf{k_p}(\mathbf{x_t} - \mathbf{x_c}) - \mathbf{k_d}(\dot{\mathbf{x}}_\mathbf{c})) + \mathbf{b} + \mathbf{g}] \\ + \mathbf{S_f}[\mathbf{F_t} + \mathbf{k_f}(\mathbf{F_t} - \mathbf{F_c})])\end{aligned} \tag{7}$$

  where $\mathbf{S_m}$ and $\mathbf{S_f}$ are Boolean diagonal selection matrices that specify which axes use motion control and which use force control ($\mathbb{R}^{6\times6}$). Although common, the axes need not be complementary.

E. Reinforcement Learning

Table IX shows PPO parameters for **Pick**, **Place**, and **Screw**. **Table X** shows a comparison of controller selection and gains on the **Screw** task. **Table XI** shows a comparison of action spaces on the **Screw** task. **Table XII** shows a comparison of baseline rewards on the **Screw** task.

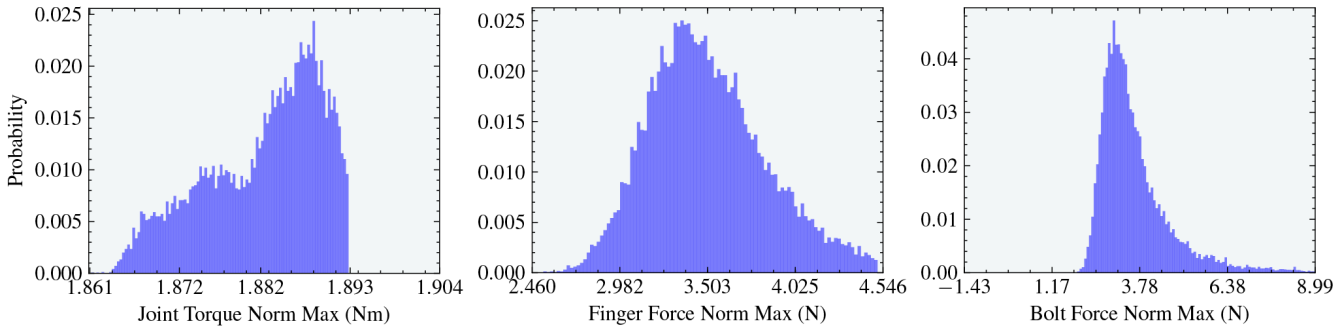

Fig. S17. Histograms of joint torques, gripper forces, nut forces, bolt forces. Data was collected during inference on the final trained **Screw** subpolicy over 128 environments and 1024 episodes. At each timestep, norms were computed for each environment, and the maximum value was recorded. Histograms show distributions of these values over all timesteps. Outliers were rejected for visualization.

	Stable Timesteps		Memory Bandwidth		Parallelization
Solver type	**Substeps**	**Iterations**	**Per frame**	**Per second**	**Max nuts/bolts**
Jacobi (before)	8	64	1.28 GB	76.8 GB	20
Jacobi (after)	8	64	24 MB	1.44 GB	1100
Gauss Seidel (before)	1	16	40 MB	2.4 GB	666
Gauss Seidel (after)	1	16	768 KB	46 MB	35000

TABLE V. Comparison of memory bandwidth requirements for a Jacobi and Gauss-Seidel solver on the nut-and-bolt scene, before and after contact reduction. *Max nuts/bolt* denotes the theoretical maximum number of nuts and bolts that could be simulated on a state-of-the-art GPU; in practice, limitations aside from contact constraint solution (*e.g.*, scaling of SDF-based contact generation) typically reduce this upper bound by one order of magnitude. Note that although *Gauss-Seidel (before)* has lower memory bandwidth requirements than *Jacobi (before)*, it is not preferred due to its low execution speed.

	Representation Info				SDF Stats		Mesh Stats	
Scene	**Envs**	**Bodies**	**SDF**	**Mesh**	**Resolution**	**Memory**	**Vertices**	**Triangles**
Peg-in-hole	1024	2	Peg	Hole	43x43x104	3.55 MB	296	592
Nut-and-bolt	1024	2	Bolt	Nut	144x404x144	31.95 MB	8899	17798
D-sub connector	1024	2	Socket	Plug	534x129x254	66.75 MB	11935	23866
Gear assembly	1024	4	Gear	Gear	314x314x104	39.1 MB	17557	35313
Nuts	1	1024	Nut	Nut	284x132x327	46.76 MB	8899	17798
Bowls	1	1024	Bowl	Bowl	257x75x257	14.36 MB	582	1160
Toruses	1	1024	Torus	Torus	304x104x304	36.60 MB	1024	6144
Franka + nut-and-bolt	128	44	Bolt	Nut	144x404x144	31.95 MB	8899	17798

TABLE VI. Geometric representations used in the simulator test scenes. *Bodies* denotes the number of bodies in each parallel environment. For *Nuts*, *Bowls*, and *Toruses*, all bodies are simulated in the same environment. *SDF* and *Mesh* denote which body in the contact pair is represented as an SDF, and which is represented as a mesh that is queried against the SDF. For *Gear assembly*, statistics are provided for SDF-mesh collisions between the large and medium gears. For *Franka + nut-and-bolt*, which contains more than 2 types of bodies, the Franka-nut collisions are handled with the Franka gripper fingers as convexes and the nut as a mesh, and the nut-nut collisions are handled with the nuts as convexes.

Part	Standards	Configurations	Clearance (mm)
4 mm round peg 4 mm round hole	ISO 286	loose, tight	0.104-0.112
8 mm round peg 8 mm round hole	ISO 286	loose, tight	0.105-0.114
12 mm round peg 12 mm round hole	ISO 286	loose, tight	0.206-0.217
16 mm round peg 16 mm round hole	ISO 286	loose, tight	0.506-0.517
4 mm rect. peg 4 mm rect. hole	ISO 286	loose, tight	0.11-0.14
8 mm rect. peg 8 mm rect. hole	ISO 286	loose, tight	0.134-0.224
12 mm rect. peg 12 mm rect. hole	ISO 286	loose, tight	0.1374-0.2274
16 mm rect. peg 16 mm rect. hole	ISO 286	loose, tight	0.1576-0.2612
M4 nut	ISO 724, 965	loose, tight	0.416-0.736
M4 bolt	ISO 724, 965	loose, tight	
M8 nut	ISO 724, 965	loose, tight	0.848-1.325
M8 bolt	ISO 724, 965	loose, tight	
M12 nut	ISO 724, 965	loose, tight	1.26-1.86
M12 bolt	ISO 724, 965	loose, tight	
M16 nut	ISO 724, 965	loose, tight	1.472-2.127
M16 bolt	ISO 724, 965	loose, tight	
M20 nut	ISO 724, 965	loose, tight	1.879-2.664
M20 bolt	ISO 724, 965	loose, tight	
Gear base, shafts Gear small Gear medium Gear large	ISO 286	loose, tight	0.005-0.014

TABLE VII. Information on peg, hole, nut, bolt, and gear models (49 parts, 14 distinct subassemblies) provided and simulated in this work. *Loose* and *Tight* configurations correspond to the ends of the manufacturing tolerance band specified by the listed standard. *Clearance* designates the corresponding range of clearances between the given part and its mating component (*e.g.*, between peg and hole). Clearance values are 2-sided (*e.g.*, for a round peg, the clearance is diametral).

Part	Manufacturer	Configurations	Clearance (mm)
BNC plug (inner)	Amphenol	visual, collision	0.1778
BNC plug (outer)	Amphenol	visual, collision	
BNC socket	Molex	visual, collision	
D-sub plug	Assmann	visual, collision	0.0
D-sub socket	Assmann	visual, collision	
RJ45 plug	Harting	visual, collision	0.34177
RJ45 socket	Amphenol	visual, collision	
USB plug	Bulgin	visual, collision	0.0
USB socket	Amphenol	visual, collision	
Waterproof plug	Harting	visual, collision	0.4
Waterproof socket	Harting	visual, collision	

TABLE VIII. Information on electrical connector assets (11 parts, 5 distinct subassemblies) provided in this work. The rigid subassemblies (BNC, D-sub, USB) are also simulated; although the deformable parts (RJ45 plug, Waterproof socket) can be simulated using PhysX or Isaac Gym, they are not tested here. The BNC plug contains both an inner and outer component coupled by a linear spring. *Manufacturer* simply designates the source of the original CAD model used to generate the visual meshes and design the collision meshes; regardless of the manufacturer, the mating features (*e.g.*, pins, holes) for all parts of a certain type are highly standardized. *Clearance* denotes the minimum clearance between the given part and its mating component (*i.e.*, between plug and socket) at the beginning of insertion. Clearance values are 2-sided (*e.g.*, for the D-sub connector, the clearance is the diametral clearance between the pins and holes). The D-sub and USB connectors begin with zero-clearance, whereas the remaining components achieve zero-clearance during insertion. Links to part-specific webpages are available on our website.

Parameter	Value
MLP network size	[256, 128, 64]
Horizon length (T)	32
Adam learning rate	1.0e-4
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Entropy coefficient	0.0
Critic coefficient	2.0
Minibatch size	512
Minibatch epochs	8
Clipping parameter ϵ	0.2

TABLE IX. PPO parameters used in pick, place, and screw subpolicies. The actor and critic networks are MLPs with a shared trunk. Observations, actions, advantages, and values were all normalized. The learning rate is small and fixed, as higher returns were consistently achieved compared to an adaptive schedule based on a KL-divergence threshold.

Controller	P Motion Gains	D Motion Gains	P Force Gains	Success Rate	Time to Success	Reward	Joint Torque (Nm)
Task-space impedance	0.01	0.1	N/A	0.0	N/A	-0.321	1.993
	0.1	0.1	N/A	0.0	N/A	-0.308	1.966
	1	0.1	N/A	0.0	N/A	-0.299	1.966
OSC motion	0.1	1	N/A	**0.797**	**2335**	**-0.101**	**1.743**
	1	1	N/A	0.669	2449	-0.115	1.758
	10	1	N/A	0.0	N/A	-0.305	2.891
Hybrid force-motion	0.1	1	0.01	0.0	N/A	-0.411	4.214
	1	1	0.1	0.0	N/A	-1.399	28.179
	10	1	1	0.0	N/A	-4.274	418.957

TABLE X. Comparison of controller selection and gains on **Screw** task. *P Motion Gains* specifies the value of the proportional gain for the first 5 elements of the action space for the motion controller; in order to bias towards efficient screwing, element 6 was assigned to be 10 for task-space impedance and 100 for the others. *D Motion Gains* specifies the value of the derivative gain for all 6 elements of the action space for the motion controller. *P Force Gains* specifies the value of the proportional gain for all 3 elements of the closed-loop force controller. *Success Rate* specifies the fraction of episodes that were successful. *Time to Success* specifies the mean number of timesteps required to achieve success. *Reward* specifies the mean reward during each episode. *Joint Torque* specifies the mean joint torque norm during each episode. Each cell is computed from the average of 3 seeds.

Actions	Observations	Success Rate	Env Steps to Success	Reward	Joint Torque (Nm)
2-DOF (Z, yaw)	Pose	0.7734	**2308**	-0.1024	**1.7378**
	Pose, velocity	**0.7969**	3658	**-0.1004**	1.7397
6-DOF	Pose	0.0	N/A	-0.1602	1.7838
	Pose, velocity	0.0	N/A	-0.1647	1.8008

TABLE XI. Comparison of action spaces on performance of **Screw** task. For the 2-DOF case, no task-space force/torque was actively generated along the non-active dimensions of the action space. Given the competitive nature of the results in the observation-space assessment, observation spaces were also tested here. *Success Rate* specifies the fraction of episodes that were successful. *Time to Success* specifies the mean number of timesteps required to achieve success. *Reward* specifies the mean reward during each episode. *Joint Torque* specifies the mean joint torque norm during each episode. Each cell is computed from the average of 3 seeds.

Baseline	Success Rate	Env Steps to Success	Joint Torque (Nm)
Linear: $-\|\|d\|\|$	0.8359	3148	1.7295
Exponential: $e^{-\|\|d\|\|}$	0.0052	3033	0.7425
Inverse: $\frac{1}{1+\|\|d\|\|}$	0.0	N/A	0.7270

TABLE XII. Comparison of baseline rewards on performance of **Screw** task. Quantity $||d||$ denotes keypoint distance, defined in **Section V-D**. *Success Rate* specifies the fraction of episodes that were successful. *Time to Success* specifies the mean number of timesteps required to achieve success. *Joint Torque* specifies the mean joint torque norm during each episodes. Due to the distinct range of each baseline reward, mean reward is not provided. Each cell is computed from the average of 3 seeds.