Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

IndustReal: Transferring Contact-Rich Assembly Tasks from Simulation to Reality

Bingjie Tang[*,1], Michael A. Lin[*,2], Iretiayo Akinola[3] , Ankur Handa[3], Gaurav S. Sukhatme[1], Fabio Ramos[3,4], Dieter Fox[3,5], Yashraj Narang[3]
[*]Equal Contribution
[1]University of Southern California, [2]Stanford University, [3]NVIDIA Corporation,
[4]University of Sydney, [5]University of Washington

***Abstract*—Robotic assembly is a longstanding challenge, requiring contact-rich interaction and high precision and accuracy. Many applications also require adaptivity to diverse parts, poses, and environments, as well as low cycle times. In other areas of robotics, simulation is a powerful tool to develop algorithms, generate datasets, and train agents. However, simulation has had a more limited impact on assembly. We present IndustReal, a set of algorithms, systems, and tools that solve assembly tasks in simulation with reinforcement learning (RL) and successfully achieve policy transfer to the real world. Specifically, we propose 1) simulation-aware policy updates, 2) signed-distance-field rewards, and 3) sampling-based curricula for robotic RL agents. We use these algorithms to enable robots to solve contact-rich pick, place, and insertion tasks in simulation. We then propose 4) a policy-level action integrator to minimize error at policy deployment time. We build and demonstrate a real-world robotic assembly system that uses the trained policies and action integrator to achieve repeatable performance in the real world. Finally, we present hardware and software tools that allow other researchers to reproduce our system and results. For videos and additional details, please see our project website.**

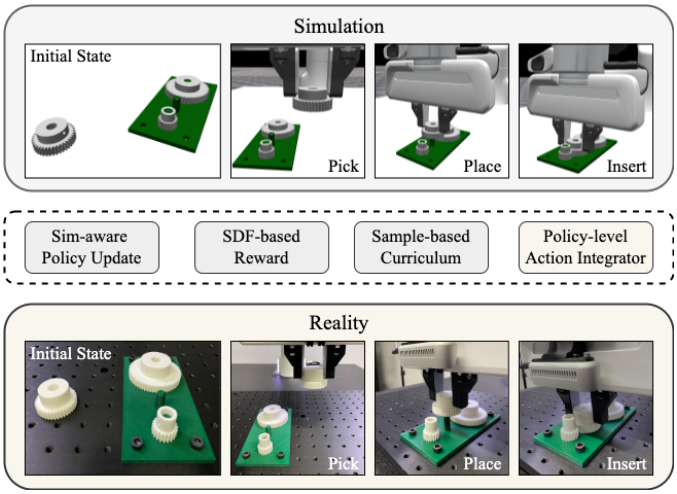


Fig. 1: **Overview.** Top: Simulation-based policy learning for one of our tasks, gear assembly. Middle: Proposed algorithms to facilitate sim-based learning and real-world deployment. Bottom: Successful transfer to the real world.

I. Introduction

Robotic assembly is a longstanding challenge [70, 26]. Assembly requires contact-rich interactions and high precision and accuracy; high-mix, low-volume settings also require adaptivity to diverse parts, poses, and environments. Today, robotic assembly is ubiquitous in the automotive, aerospace, and electronics industries. However, assembly robots are typically expensive, achieving precision primarily through hardware rather than intelligence. Moreover, systems often require meticulous engineering of adapters, fixtures, lighting, and robot trajectories. Such efforts demand substantial time and effort from robotics integrators and can result in solutions that are highly sensitive to perturbations of the robotic workcell.

Simulation is an indispensable means to solve engineering challenges. For example, simulators are used for finite element analysis, computational fluid dynamics, and integrated circuit design. Nevertheless, simulation has had a comparably limited impact on robotic assembly. Accurate simulation of geometrically-complex parts can require generating $1k$-$10k$ contacts per rigid body pair, followed by solving a nonlinear complementarity problem (NCP) at each contact. Furthermore, robotic assembly tasks require long-horizon sequential decision-making; powerful data-driven methods (e.g., on-policy RL) have high sample complexity, requiring faster-than-realtime simulation. Achieving such accuracy and performance requirements has only recently become possible [41, 48, 29].

Given modeling limitations and finite compute, simulation will always differ from reality; this reality gap has been notoriously large for robotics. Sim-to-real transfer methods address this gap through techniques such as system identification and domain randomization. These methods have shown remarkable results in locomotion [30, 51, 44] and manipulation [5, 18, 9]. However, sim-to-real efforts for assembly have been scarce.

We present **IndustReal**, a set of algorithms, systems, and tools for solving contact-rich assembly tasks in simulation and transferring behaviors to reality (Figure 1). Specifically, our primary contributions are the following:

- **Algorithms**: For simulation, we propose three methods to allow RL agents to solve contact-rich tasks in a simulator: a **simulation-aware policy update (SAPU)** to reward the agent when simulation predictions are more reliable, a **signed distance field (SDF)-based dense reward** to provide an alignment metric between geometrically-complex objects, and a **sampling-based curriculum (SBC)** to prevent overfitting to earlier stages of a curriculum. For sim-to-real transfer, we also propose a **policy-level action integrator (PLAI)**, which reduces steady-state error in

the presence of unmodeled dynamics (e.g., friction).

- **Benchmarks**: We solve several challenging tasks proposed in Factory [48] (pick, place, and insertion tasks for pegs-and-holes and gear assemblies in simulation) with success rates of 82-99%. We provide careful evaluations over 265k simulated trials to show the utility of SAPU, SDF-based rewards, and SBC for solving these tasks.
- **Systems**: We design and demonstrate a real-world system that can perform sim-to-real transfer of our simulation-trained policies, with success rates of 83-99% over 600 trials. We provide careful evaluations to show the utility of PLAI. To our knowledge, this is the first system for sim-to-real of all phases of the assembly problem: from detection, to grasping, to part alignment, to insertion. Our system uses commonly-used robotics hardware and requires no real-world policy adaptation phase.

Our secondary contributions are the following:

- **Hardware**: We present **IndustRealKit**, which contains CAD models for all parts designed for our setup, as well as a list of all purchased parts. The CAD models can all be printed on a desktop 3D printer. IndustRealKit allows the research community to easily replicate our experimental hardware and benchmark their performance.
- **Software**: We present **IndustRealLib**, a lightweight Python library that allows users to easily deploy policies trained in NVIDIA Isaac Gym [43] onto a real-world Franka Emika Panda robot [17]. The library also contains code to assist with policy training. IndustRealLib allows the research community to reproduce our robot behaviors.

We aim for **IndustReal** to provide algorithms, benchmark results, and a reproducible system that serve as a path forward for sim-to-real transfer on contact-rich assembly tasks.

II. Related Work

We divide prior work on robotic assembly into three categories: 1) classical approaches leveraging analytical methods [70, 45], 2) learning-based approaches leveraging real-world data or experience [75], and 3) RL-based sim-to-real approaches leveraging robotics simulators. We defer a review of (1) and (2) to Appendix A and focus on (3).

A. Sim-to-Real Transfer for Assembly

Over the past few years, there have been a number of impressive efforts in sim-to-real for assembly. These efforts have primarily used MuJoCo [52, 11, 20, 67, 79, 27] or PyBullet [38, 56, 54]; have used PPO [56, 58, 20, 27] or DDPG [38, 6]; and have aimed to solve peg-in-hole [38, 7, 54, 52, 58, 11, 67, 79, 27] or NIST-style tasks [7, 52, 11, 79]. However, several of these studies use large clearances (e.g., $\geq$ 1 mm) and/or large parts in simulation and/or the real world. Furthermore, almost all use force/torque (F/T) sensors to collect observations and/or set thresholds. Most require human demonstrations [38, 6, 11], a baseline motion plan [56, 58, 27], and/or fine-tuning in the real-world [7, 52]. Finally, all but one [52] focus only on insertion and assume the object is pre-grasped; however, [52] also uses specialized grippers, low-dimensional action spaces, highly-constrained target locations, and a real-world policy adaptation phase.

In contrast, we make several design choices that increase the realism of the problem and encourage reproducibility. First, for software, we use Factory [48] within Isaac Gym, which can solve contact dynamics between highly-complex geometries without simplification. Second, for hardware, we use a collaborative robot (Franka Panda) and RGB-D camera (Intel RealSense D435) that are widespread in research, but far less precise than those in industrial assembly. We use no task-specific grippers and avoid F/T sensors due to their cost, noise, and fragility. We also use realistic part clearances ($\leq$0.5-0.6 mm, aligned with the upper bound of ISO 286). Next, for problem scope, we address sim-to-real for all parts of the assembly sequence (i.e., detection, grasping, alignment, and insertion). We face robustness challenges due to calibration and localization error; moreover, we apply large randomizations of part poses and targets. Finally, in methodology, we achieve sim-to-real transfer without baseline plans or demos, dynamics randomization, or real-world policy adaptation phases.

III. Problem Description

A. Problem Setup

Our problem setup is as follows: a Franka robot is mounted to a work surface. A RealSense D435 camera is mounted to the robot wrist. Industrial-style parts inspired by the NIST Task Board 1 [25, 26] are placed upright on the work surface. The parts with extruded features (which we henceforth refer to as *plugs*) have a randomized 3-DOF pose (x, y, θ) on top of an optical breadboard and are free to move; the parts with mating features (which we call *sockets*) also have a randomized pose, but are bolted to the breadboard to emulate industrial fixturing.[1] The fundamental task is to perceive, grasp, transport, and insert all the plugs into their corresponding sockets.

Specifically, we aim to perform this task for three types of assemblies from [48] (Figure 2 column 1):

- **Pegs and Holes:** 2 different classes of pegs (round and rectangular), each with 3 different sizes (max dimension: 8 mm, 12 mm, and 16 mm) must be inserted into corresponding holes (clearances: 0.5-0.6 mm).
- **Gears and Gearshafts:** 3 different gears (diameters: 20 mm, 40 mm, 60 mm) must be inserted onto corresponding gearshafts (diametral clearances: 0.5 mm).
- **Connectors and Receptacles:** 2 different connectors (2-prong NEMA 1-15P and 3-prong NEMA 5-15P) must be inserted into corresponding receptacles.

We first aim to solve the assembly task in simulation using RL with CAD models of the robot and the objects. We then aim to transfer the policies to the real world.

B. Problem Decomposition

For each category of parts, to facilitate the assembly task, we decompose it into three phases (Figure 2 columns 2-4):

[1]This broad usage of *plug* and *socket* will persist throughout the paper.

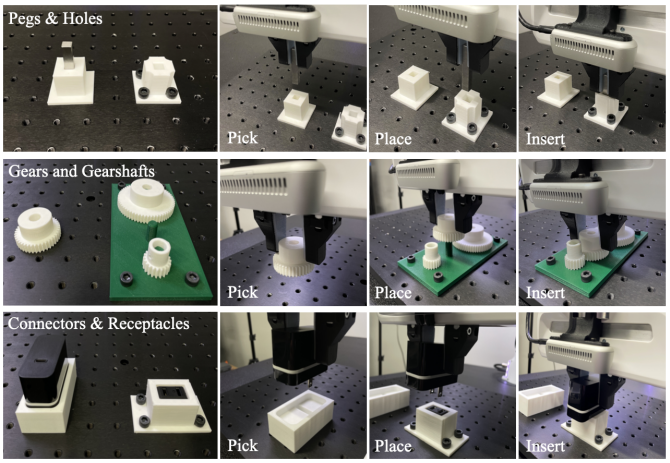


Fig. 2: **Problem setup and decomposition.** Column 1: Three types of assemblies. Columns 2-4: Goal states of Pick, Place, and Insert phases.

- **Pick**: The robot grasps a randomly-positioned plug (i.e., peg, gear, or connector) within its workspace.
- **Place**: The robot transports the grasped plug close to its corresponding socket (i.e., hole, gearshaft, or receptacle).
- **Insert**: The robot brings the grasped plug into contact with its socket and inserts the plug, aligning parts where necessary (e.g., when inserting intermediate gears).

IV. Policy Learning in Simulation

In this section, we first describe our general strategies for policy learning in Sections IV-A-IV-D. Although these strategies were sufficient to train successful **Pick** and **Place** policies, they were inadequate for training **Insert** policies ($\approx 12\%$ success rates), motivating our algorithmic development. We describe our 3 simulation-based algorithms *and evaluate them* on our **Insert** policies in Sections IV-E-IV-G[2].

A. Training Environments

We developed our code within the Factory simulation framework [48]. For the **Peg and Hole** assemblies, we first trained a **Reach** policy, where the robot learned to move its end-effector to a randomized pose within a large workspace. We then fine-tuned **Reach** to solve the **Pick** task by including all peg assets in the scene and redefining success as lifting the pegs. Similarly, we fine-tuned **Reach** to solve the **Place** task by initializing the pegs within the robot grippers, including all hole assets in the scene, and redefining success as bringing the pegs to their corresponding holes. Empirically, for the **Pick** and **Place** tasks, training in free space and fine-tuning on contact was more efficient than training from scratch. However, we trained an **Insert** policy from scratch.

For the **Gears and Gearshafts** assemblies, we did not train policies to solve the **Pick** or **Place** tasks; as we later show, we solved those tasks in the real world by executing the corresponding **Peg and Hole** policies, demonstrating generalization. However, we again trained an **Insert** policy from scratch.

[2]Note that when we evaluate each algorithm, the other two algorithms are used; thus, the evaluations function as ablation studies.

Finally, for the **Connectors and Receptacles** assemblies, we did not train policies for any phase. Again, we later show that we solved those tasks in the real world via policy transfer.

In summary, for the **Peg and Hole** assemblies, we trained **Pick**, **Place**, and **Insert** policies, and for the **Gears and Gearshafts** assemblies, we trained another **Insert** policy.

B. Formulation

We formulated the problem as a Markov decision process (MDP) with state space $\mathcal{S}$, observation space $\mathcal{O}$, action space $\mathcal{A}$, state transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, initial state distribution ρ_0, reward function $r : \mathcal{S} \rightarrow \mathbb{R}$, horizon length T, and discount factor $\gamma \in (0, 1]$. The objective was to learn a policy $\pi : \mathcal{O} \rightarrow \mathbb{P}(\mathcal{A})$ that maximized the expected sum of discounted rewards $\mathbb{E}_\pi[\Sigma_{t=0}^{T-1}\gamma^t r(s_t)]$.

We used proximal policy optimization (PPO) [53] to learn a stochastic policy $a \sim \pi_\theta(o)$ (actor), mapping from observations $o \in \mathcal{O}$ to actions $a \in \mathcal{A}$ and parameterized by a network with weights θ; as well as an approximation of the on-policy value function $v = V_\phi(s)$ (critic), mapping from states $s \in \mathcal{S}$ to value $v \in \mathcal{V}$ and parameterized by weights ϕ. We used the PPO implementation from *rl-games* [42]; hyperparameters and architectures are in Table XI. Finally, we aimed to train the policy in simulation and deploy in the real world with no policy adaptation phase on the specific real environment.

C. Observations, Actions, and Rewards

Our observation spaces in simulation and the real world were task-dependent. The observations provided to the actor consisted exclusively of joint angles, gripper/object poses, and/or target poses, as the Franka's joint velocities and joint torques exhibited appreciable noise in the real world. However, we employed asymmetric actor-critic [50], where velocity information was still used to train the critic. Our exact observations for all policies are listed in Table V.

Our action spaces for both simulation and the real world were task-independent. The actions consisted of incremental pose targets to a task-space impedance (TSI) controller (specifically, $a = [\Delta x; \Delta q]$, where Δx is a position error and Δq is a quaternion error). We learned incremental targets rather than absolute targets because the latter encodes task-specific biases and must be selected from a large spatial range. We used TSI rather than operational-space control (OSC) because Franka provides a high-performance implementation of TSI, and OSC relies on an accurate dynamics model.

Our rewards in simulation were task-dependent. However, all rewards could be expressed in the following general form:

$$G = w_{h_0}..w_{h_m}\left(\sum_{t=0}^{H-1}[w_{d_0}R_{d_0}(t) + ... + w_{d_n}R_{d_n}(t)] + w_{s_0}R_{s_0} + ... + w_{s_p}R_{s_p}\right) \quad (1)$$

where G is the return over the horizon, $R_{d0}...R_{dn}$ are distinct dense rewards, H is the horizon length, $R_{s_0}...R_{s_p}$ are terminal success bonuses, $w_{d_0}...w_{d_n}$ and $w_{s_0}...w_{s_p}$ are scaling factors

that map distinct rewards into a consistent unit system and weight the importance of each term, and $w_{h_0}...w_{h_m}$ are scaling factors on the return over the entire horizon. Not all terms are used in each phase, and most of our reward formulations are simple. Detailed formulations and success criteria are provided in Table VI and Table IV, respectively.

D. Randomization and Noise

At the start of each episode, we randomized the 6-DOF end-effector and object poses over a large spatial range. In addition, for the **Insert** policies, we introduced observation noise. As well established, these perturbations are critical for ensuring robustness to initial conditions and sensor noise in the real world. Randomization and noise ranges are provided in Table VII). However, we avoided dynamics randomization [49, 18], as we had strong priors on our system dynamics.

E. Simulation-Aware Policy Update (SAPU)

Method: In contact-rich simulators, spurious interpenetrations between assets are unavoidable, especially when executing in real-time (Figure S15). Unfortunately, in simulation for RL, an agent can exploit inaccurate collision dynamics to maximize reward, learning policies that are unlikely to transfer to the real world [47]. Thus, we propose our first algorithm, a **simulation-aware policy update (SAPU)**, where the agent is encouraged to learn policies that avoid interpenetrations.

Specifically, we implemented a GPU-based interpenetration-checking module using *warp* [40]. For a given environment, the module takes as input the plug and socket mesh and associated 6-DOF poses. The module samples $N = 1000$ points on/inside the mesh of the plug, transforms the points to the socket frame, computes distances to the socket mesh, and returns the max interpenetration depth (algorithm 1). This procedure is performed each episode, and the depth is used to weight the cumulative reward during the policy update.

Evaluation: We evaluate **SAPU** on the **Peg and Hole** assembly **Insert** policy, with the following test cases:

- **Baseline:** Do not utilize interpenetration information.
- **Filter Only:** For a given episode, if max interpenetration depth d_{ip}^{max} is greater than $\epsilon_{ip} = 1\ mm$, do not use return in policy update. If $d_{ip}^{max} < \epsilon_{ip}$, use return as normal.
- **Weight Only:** For a given episode, weight return by $1 - \tanh(d_{ip}^{max}/\epsilon_d)$, which is bounded by $(0, 1]$.
- **Filter and Weight:** For a given episode, if $d_{ip}^{max} > \epsilon_{ip}$, do not use return in policy update. If $d_{ip}^{max} < \epsilon_{ip}$, weight return by $1 - \tanh(d_{ip}^{max}/\epsilon_d)$).

After training policies with each strategy, we tested each policy in simulation over 5 seeds, with 1000 trials per seed; quantified d_{ip}^{max} for each episode; and evaluated success rate over all episodes (Figure 3). Success was defined as inserting the peg into the hole (Table IV); however, to also ascertain whether success was achieved in the desired way (i.e., by avoiding interpenetration), we computed success rate for episodes where $d_{ip}^{max} < [0.5, 1, 1.5, 2, \infty]$ mm.

For both the most realistic scenario (when only successes where $d_{ip}^{max} < 0.5$ mm were counted) and the least realistic scenario (when all successes were counted), the **Baseline**, **Filter Only**, and **Weight Only** strategies were unable to achieve success rates above 40%. However, the **Filter and Weight** strategy performed well over all scenarios, with success rates of 84.9- 87.6%. Thus, **Filter and Weight** was not only most effective at learning a policy that avoided interpenetration, but was also most effective at policy learning *in general*.

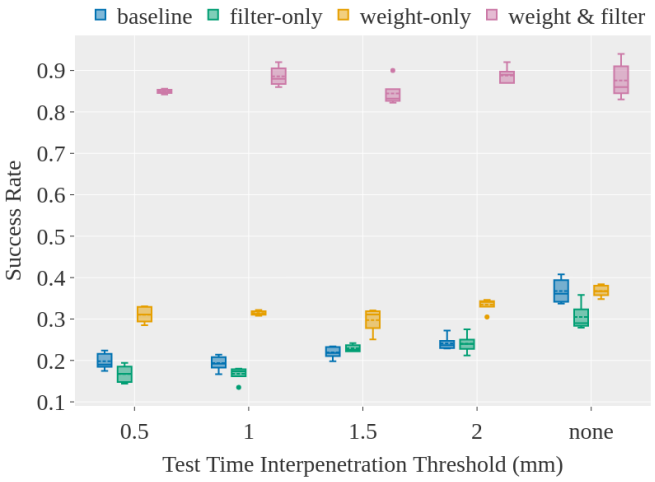


Fig. 3: **Evaluation of Simulation-Aware Policy Update**. Success rates are computed for episodes where the maximum interpenetration distance was less than the specified value at test time. Boxes indicate median and IQR.

Algorithm 1: Interpenetration Checking Per Env.

Input: plug mesh m_p, socket mesh m_s, plug pose p_p, socket pose p_s, number of query points N.

sample N points in $m_p \rightarrow \mathbf{v}$, $\mathbf{v} = \{v_0, ..., v_{N-1}\}$;
transform $\mathbf{v}$ to current m_p pose p_p in m_s frame ;
for *every vertex* $v \in \mathbf{v}$ **do**
    compute closest point on m_s to v;
    if v *inside* m_s **then**
        calculate interpenetration distance;
d_{ip}^{max} = max interpenetration from all $v \in \mathbf{v}$ to m_s;
return d_{ip}^{max};

F. SDF-Based Dense Reward

Method: Keypoint-based rewards are widely used, as they avoid weighting between distinct position and orientation rewards [3]. However, collinear keypoints (e.g., [48]) underspecify the assembly of non-axisymmetric parts (e.g., rectangular pegs), and non-collinear keypoints overspecify the assembly of symmetric parts, as identical geometries do not alias. Thus, we propose a **signed distance field (SDF)-based dense reward**, where an SDF is defined as a map $\phi(\boldsymbol{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ from an arbitrary point $\boldsymbol{x}$ to its signed distance $\phi(\boldsymbol{x})$ to a surface.

Specifically, for each plug mesh in its nominal pose, we use sampling to preselect $N = 1000$ points on the surface. In addition, for the same mesh in its *target pose*, we use *pysdf* to precompute and store the SDF values at each cell of a

dense voxel grid containing the mesh. During training, the preselected points are transformed to the frame of the plug mesh in its *current pose*, and the SDF values are queried at these points. This procedure is performed at each timestep in each environment and is used to generate a reward signal.

Evaluation: We evaluate **SDF-Based Dense Reward** on the **Pegs and Holes** assembly **Insert** policy by comparing the following reward formulations (Table I):

- **Collinear Keypoints:** Each object has 4 keypoints along its Z-axis; Euclidean distances are summed and averaged.
- **6-DOF Keypoints:** Each object has 13 keypoints over 3 axes; Euclidean distances are summed and averaged.
- **Chamfer Distance:** Each object has a point cloud defined by its mesh vertices; chamfer distance [13] is computed.
- **SDF Query Distance:** The root-mean-square SDF distance is computed as described earlier.

After training policies with each strategy, we tested each policy in simulation over 5 seeds, with 1000 trials per seed; quantified terminal position and rotation error for each episode; and quantified success rate and engagement rate (Table I). Success was defined as inserting the peg into the hole; *engagement* was defined as a partial insertion.

The **Collinear Keypoints**, **6-DOF Keypoints**, and **Chamfer Distance** rewards resulted in appreciable position and rotation errors of 11.7-31.0 mm and 0.13-0.99 rad, respectively, with chamfer distance performing the worst; as follows, success rates varied between 1.8-54.2%. However, the **SDF Query Distance** reward resulted in errors of just 3.80 mm and 0.086 rad, with a success rate of 88.6% and near-perfect engagement rate of 96.6%. Thus, **SDF Query Distance** was by far the most effective reward formulation for policy learning. As Factory [48] already precomputes SDFs for all objects for contact generation, we envision a single representation-generation step for both physics and reward.

G. Sampling-Based Curriculum (SBC)

Method: Curriculum learning [8] is an established approach for solving long-horizon problems; as the agent learns, the difficulty of the task is gradually increased. Nevertheless, for both the **Peg and Hole** and **Gears and Gearshafts** assemblies, for the **Insert** phase, naive implementations of curriculum learning (i.e., increasing initial distance from goal) were ineffective; when the initial peg/gear state was above the hole/gearshafts, the agent failed to progress, likely overfitting to a partially-inserted plug. Thus, we developed **Sampling-Based Curriculum (SBC)**, whereby the agent is exposed to the entire range of initial state distributions from the start of the curriculum, but the lower bound is increased at each stage.

Specifically, let z^{low} denote the lower bound of the initial height of a plug above its socket at a given curriculum stage, and let z^{high} denote a constant upper bound; the initial height of the plug is uniformly sampled from $\mathrm{Uniform}[z^{low}, z^{high}]$. In addition, let Δz^i and Δz^d denote an increase or decrease in z^{low}, and let p_n denote the mean success rate over all environments during episode n. When episode n terminates, we update z^{low} as follows:

$$z^{low} \leftarrow \begin{cases} z^{low} + \Delta z^i, & p_n > 80\% \\ z^{low} - \Delta z^d, & p_n < 10\% \\ z^{low}, & \text{otherwise.} \end{cases}$$

In general, we enforce $\Delta z^d < \Delta z^i$. We define an increase in z^{low} as an advance to the next stage of the curriculum, and a decrease in z^{low} as a reversion to the previous stage.

Evaluation: We evaluate **SBC** on the **Pegs and Holes** assembly **Insert** phase, with the following test cases:

- **Baseline:** No curriculum learning is used; $z^{init} = z^{high}$.
- **Standard:** Peg height is initialized at z^{low}; at each stage, z^{low} increases, until a max value of z^{high}.
- **Sampling-Based:** Initial peg height is sampled as described earlier.

For the **Standard** and **Sampling-Based** strategies, z_{low} was initially 10 mm below the top of the hole, and z_{high} remained constant at 10 mm above. The criterion for advancing to the next stage was an 80% success rate. *Engagement* was defined as partially inserting the peg into the hole; success was defined as full insertion. We set $\Delta z^i = 5mm$ and $\Delta z^d = 3mm$. Whenever the peg was initialized above the hole, we also perturbed its position along the X- and Y-axes.

After training policies with each strategy, we tested each policy in simulation over 5 seeds, with 1000 trials per seed (Table II). All strategies achieved moderate rotation errors of 0.086-0.096 rad, and the **Baseline** and **Sampling-Based** strategies both achieved high engagement rates of 89.2-96.6%. However, **Sampling-Based** substantially outperformed the others in success rate and position error; it achieved 88.6% and 3.80 mm, respectively, whereas the others performed no better than 66.8% and 10.7 mm. These results substantiate existing evidence that curriculums can facilitate RL when carefully implemented, and importantly, suggest a specific implementation in the case of discontinuous contact.

H. Joint Evaluation

As described in Sections IV-E-IV-G, we proposed three algorithms for improving learning of contact-rich **Insert** policies: **Simulation-Aware Policy Update** to adapt to simulator inaccuracy, **SDF-Based Dense Reward** to quantify alignment for asymmetric or symmetric objects, and **Sampling-Based Curriculum** to prevent overfitting to initial partial insertions. As a final evaluation, we comprehensively evaluated all three techniques in tandem (Figure 4 and Table X). When training and testing with moderate state randomization (plug/hole randomization of $\pm$10 mm/$\pm$10 cm, respectively) and observation noise ($\pm$1 mm), the **Pegs and Holes** assembly **Insert** policy achieved success and engagement rates of 88.6% and 96.6%, respectively, whereas the **Gears and Gearshafts** assembly **Insert** policy achieved 82.0% and 85.2%. Across all evaluations, worst-case performance was 67.88% (when testing with *twice* the gearshaft position randomization of training), and best-case was 92.4% (testing with no randomization or noise).

Reward	Formulation	Num. Trials	Success (%)	Engage (%)	Pos. Error (mm)	Rot. Error (rad.)
Collinear keypoints	$-\|\|\mathbf{k}_{\text{curr}} - \mathbf{k}_{\text{goal}}\|\|^2$	1000	15.40 ± 5.22	64.40 ± 3.05	16.28 ± 1.08	0.150 ± 0.020
6-DOF keypoints	$-\|\|\mathbf{k}^{6D}_{\text{curr}} - \mathbf{k}^{6D}_{\text{goal}}\|\|^2$	1000	54.2 ± 7.56	83.80 ± 4.44	11.74 ± 1.92	0.132 ± 0.013
Chamfer distance	$-\text{Chamfer\_dist}(\text{S}_{\text{plug}}, \text{S}_{\text{socket}})$	1000	1.80 ± 1.92	47.80 ± 6.14	31.02 ± 0.95	0.994 ± 0.030
SDF query distance	$-\log(\Sigma_i^N \phi(x_i)/N)$	1000	88.60 ± 2.41	96.60 ± 2.30	3.80 ± 0.80	0.086 ± 0.026

TABLE I: **Evaluation of SDF-Based Dense Reward.** Symbol $\mathbf{k}$ denotes object keypoint positions, S is a set of points comprising a point cloud (here, we use plug/socket mesh vertices), and x_i denotes points sampled from the plug mesh (again, we use vertices). *Engage* denotes a partial insertion.

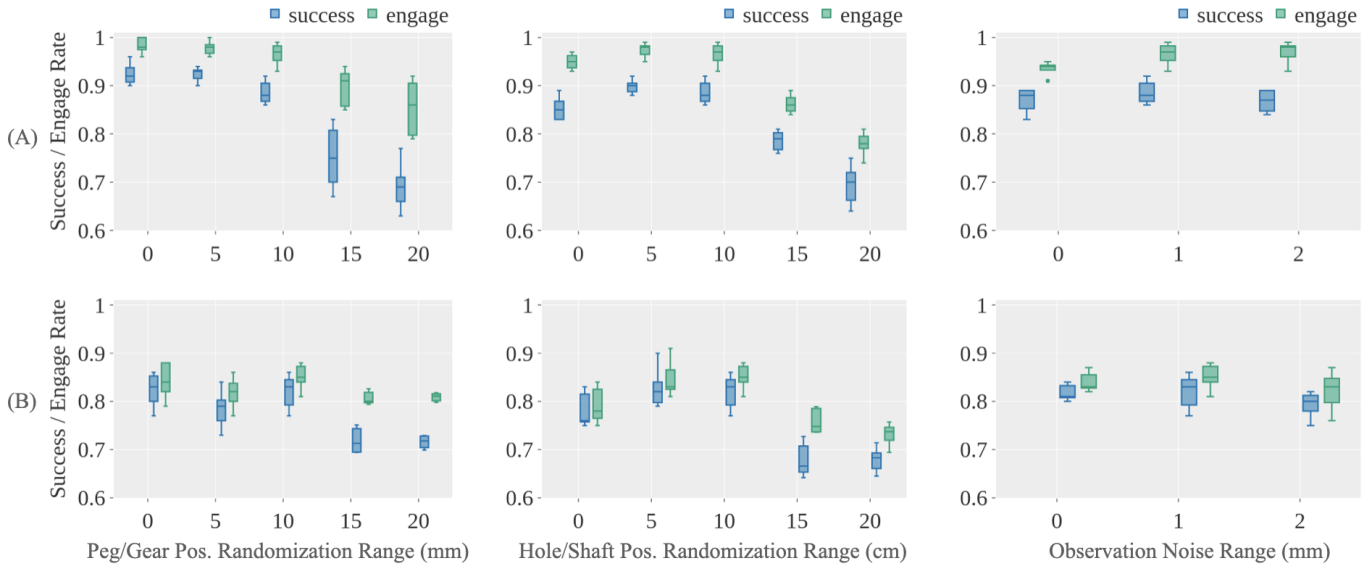


Fig. 4: **Joint evaluation of Simulation-Based Policy Update, SDF-Based Dense Reward, and Sampling-Based Curriculum.** (A) **Pegs and Holes** assembly **Insert** policy. (B) **Gears and Gearshafts** assembly **Insert** policy. *Engage* denotes partial insertion. Policies were trained with moderate randomization (plug/socket randomization of ± 10 mm and 10 cm, respectively, and observation noise of ± 1 mm); thus, plots evaluate in-distribution and OOD performance.

Curriculum	**Success** (%)	**Engage** (%)	**Pos. Err.** (mm)	**Rot. Err.** (rad)
Baseline	66.80 ± 5.76	89.2 ± 4.97	10.70 ± 2.70	0.086 ± 0.095
Standard	32.40 ± 1.82	46.0 ± 3.61	18.16 ± 0.36	0.096 ± 0.0091
Sampling-Based	88.60 ± 2.41	96.6 ± 2.30	3.80 ± 0.80	0.086 ± 0.026

TABLE II: **Evaluation of Sampling-Based Curriculum.** *Baseline* denotes that no curriculum was used.

V. Policy Deployment in Real World

In this section, we first describe our general strategies for policy deployment in the real world in Sections V-A-V-C. We then describe *and evaluate* our deployment-time algorithm, the **Policy-Level Action Integrator (PLAI)**, in Section V-D. Finally, we provide comprehensive evaluations and demonstrations of our full real-world system in Section VI.

A. Communications Framework

We illustrate our communications stack in Figure S8. In summary, we developed the **IndustRealLib** library, which accepts trained policy checkpoints from Isaac Gym as input, and outputs targets for a Franka robot controlled via a task-space impedance (TSI) controller. The targets are sent to the *frankapy* Python library, which streams the commands via ROS to the *franka-interface* C++ library [78]. *franka-interface* then sends the commands to the *libfranka* library provided by Franka, which maps the TSI commands to low-level joint torques that are streamed to the robot. Proprioceptive signals are then communicated to **IndustRealLib** in reverse order.

Latencies of our real-world and simulated systems are summarized in Table VIII. In the real world, physics frequency is approx. infinite, low-level control frequencies (between *libfranka* and robot) are 1 kHz, and policy control frequencies (between *libfranka* and **IndustRealLib**) are 60-100 Hz (limited by inference and ROS). We thus set our physics frequency during training to the highest practical rate (120 Hz) given our compute, and restricted our control rate during training to 60 Hz to prevent aliasing of policy signals during deployment.

B. Perception Pipeline

The primary goal of our perception pipeline is to estimate the 2D poses (x, y, θ) of the parts in the robot frame. Our pipeline consists of 3 separate steps: a one-time camera calibration, a per-experiment workspace mapping, and a per-experiment object detection from a single RGB image. Bounding box centroids and a trivially-specified height are used to construct targets. All details are provided in Appendix B2-B5.

C. Dynamics Strategy

Our policies initially exhibited substantial steady-state error during deployment. We note that a partial resolution was

carefully eliminating all arbitrary energy dissipation (e.g., heuristically-applied friction and damping) in the simulator and asset descriptions; dissipative terms are often introduced to facilitate simulation stability, but their values are typically chosen without physical consideration. Simulation and real-world results with/without heuristic damping are in Figure S9.

D. Policy-Level Action Integrator

Method: Robotics simulations can exhibit marked discrepancies with the real world due to incomplete models, inaccurate parameters, and numerical artifacts [14]; although dynamics randomization can improve sim-to-real transfer, it can require substantial training time and effort [2, 5, 18] and may penalize precision. Inspired by classical PID control, which can minimize steady-state error and reject disturbances on linear systems, we propose a **Policy-Level Action Integrator (PLAI)**, which integrates policy actions during an episode.

An established approach for applying policy actions is

$$s^d_{t+1} = s_t \oplus a_t = s_t \oplus \Pi(o_t), \tag{2}$$

where s^d_{t+1} is the desired state, a_t is an action expressed as an incremental state target, s_t is the current state, o_t is the current observation, Π is the policy, and $\oplus$ computes the state update (e.g., for states defined by position and orientation, $\oplus$ computes composition with a translation and rotation).

In contrast, **PLAI** applies policy actions as

$$s^d_{t+1} = s^d_t \oplus a_t = s^d_t \oplus \Pi(o_t) \tag{3}$$

Thus, the policy action is applied to the *last desired state* instead of the current state. Unrolling from $t = 0...T$,

$$s^d_T = s^d_0 \oplus \sum_{i=0}^{T-1} a_i = s^d_0 \oplus \sum_{i=0}^{T-1} \Pi(o_i). \tag{4}$$

where s^d_0 is set to s_0[3]. Thus, the desired state at time T is equal to the initial state composed with successive actions over time, effectively integrating them. (Note that this formulation is *not* open-loop control, as the policy continues to be evaluated on the current observation o_t when generating actions.) When coupled with a low-level PD controller (e.g., a TSI controller), **PLAI** has a close relationship with a standard (non-integrating) policy coupled with a PID controller; a derivation is provided in Appendix C. Empirically, **PLAI** requires minimal implementation effort (1-2 lines of code), is simple to tune, and outperforms standard PID in our application.

Like PID controllers, **PLAI** can experience *windup* when in contact with the environment; unbounded error accumulates, resulting in unstable dynamics. To mitigate this effect, we also develop **Leaky PLAI**, which clamps the accumulated control effort. Equations are derived in Appendix C.

Evaluation: From careful observations, discrepancies in simulated and real-world dynamics are primarily caused by nonlinear friction and imperfect gravity compensation on the real robot. Thus, we trained a **Reach** policy under ideal conditions in simulation, and evaluated the ability of **PLAI** to reject friction and gravity disturbances in simulation and reality at test time. We evaluated three test-time methods:

- **Nominal:** Actions are applied to the current state.
- **PID:** Actions are applied to the current state. PID is used with classical anti-windup for best-case performance.
- **PLAI:** Actions are applied to the *desired* state.

We evaluated each method under three test-time conditions:

- **Ideal:** No friction or gravity perturbations are applied.
- **Friction:** Joint friction of 0.15 Nm is applied to all robot joints (within the range of identified values from [16]).
- **Gravity:** A gravitational perturbation of 0.12 m/s^2 is applied to all robot links.

We randomized the initial pose and target pose of the robot and measured steady-state position error (Figure 5). Notably, **PLAI** had substantially lower error and variance than **Nominal** under friction and gravity, had consistently lower error than PID, and maintained $\approx 2\,mm$ error across all test conditions.

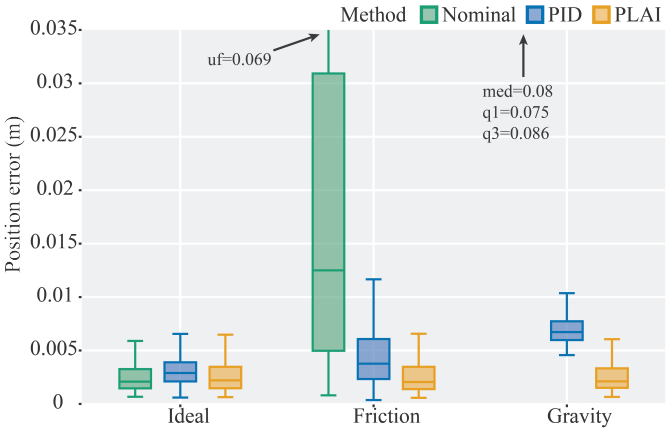


Fig. 5: **Evaluation of PLAI in simulation.** Results of **Nominal** are annotated when outside of plot bounds. Full-axis plot is in Figure S13.

Next, we conducted a similar experiment in the real world. The **Reach** policy was deployed with and without **PLAI** (Figure 6); friction and gravity perturbations were simply from real-world dynamics. Again, **PLAI** demonstrated substantially lower error and variance than **Nominal**, with $\approx 2\,mm$ error. As a final comparison, the TSI implementation provided by Franka (RL-free) was deployed on the same task and resulted in $4.45\,mm$ error. Thus, **PLAI** is a simple but highly effective means to minimize error with respect to policy targets; moreover, it can be applied exclusively at deployment time.

VI. Real-World Experiments

After developing and validating our algorithms, we performed comprehensive experiments and demos to evaluate our real-world system (Figure S10). Five types were executed: **Pick**, **Place**, **Sort**, **Insert**, and **Pick-Place-Insert (PPI)**.

A. Pick Experiment

This experiment evaluated the ability of the real-world system to initiate contact and pick up arbitrarily-placed objects.

[3]The summation symbol in Equation 4 is used as shorthand for successive compositions with actions.

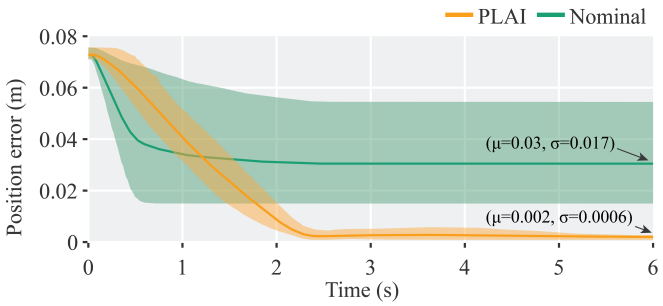


Fig. 6: **Evaluation of PLAI in the real world.** Each method was tested on 3 different goals with 20 trials each. Evaluation parameters are in Table IX.

Experimental Setup: 6 different pegs were randomly placed on top of an optical breadboard with dimensions 450 mm x 300 mm, which itself was located within bounds of 500 mm x 350 mm. The pegs were located in trays that were free to slide; angular perturbations of ± 10 deg were applied to the trays containing non-axisymmetric pegs. The goal was for the robot to detect all the pegs and use the simulation-trained **Pick** policy to pick up the objects before releasing them.

Key Results: The system demonstrated extremely high success rates (98.8%) across all pegs (Table III). Failure cases were one missed detection of a peg, as well as one grasp of both a peg and its corresponding peg tray.

B. Place Experiment

This experiment evaluated the ability of the real-world system to accurately reach low target locations while maintaining contact with a typically-sized object in the gripper.

Experimental Setup: Eight 25 mm x 25 mm trays were randomly placed on top of the breadboard. 20 mm x 20 mm printed targets were centered on top of the trays; the targets were used to measure positional accuracy and consisted of concentric rings, each with a thickness of 2 mm. A laser was rigidly mounted to the grippers (Figure S12). The goal was for the robot to detect the trays and use the simulation-trained **Place** policy to guide the laser to the centers of the targets.

Key Results: The system demonstrated low steady-state errors, with a mean distance-to-goal of 4.23 $\pm$ 1.96 mm. The error distribution is illustrated in Figure S9b.

C. Sort Demonstration

This experiment qualitatively demonstrated the ability of the robot to execute a realistic sorting procedure.

Experimental Setup: 6 different pegs and 3 different gears were randomly placed on top of the breadboard. The pegs were located in trays, and angular perturbations of ± 45 deg were applied. Bins were placed at approximately-determined positions in the workspace. The goal was for the robot to use its **Pick** and **Place** policies to detect, pick, place, and drop the round pegs, rectangular pegs, and gears into separate bins.

Key Results: Performance was highly repeatable in practice; please see the supplementary video.

D. Insert Experiment

This experiment evaluated the ability of the real-world system to insert diverse plugs into corresponding sockets, as well as generalize to unseen assets.

Experimental Setup: 6 different pegs, 3 different gears, and 2 different *unseen* NEMA connectors (2- and 3-prong) were placed imprecisely in the gripper fingers. Holes, gearshafts, and receptacles were mounted to the breadboard. The end-effector was manually guided until the plugs were inserted into their respective sockets; the end-effector pose was recorded as a target. The end-effector was then commanded to a random initial state (Table XII). The robot received an observation of the target with random X- and Y-axis noise $\sim U[-2, 2]$ mm.[4] The goal was for the robot to use its **Insert** policies to insert the plugs into their corresponding sockets.

Key Results: The system demonstrated high engagement rates (i.e., partial insertions) across the pegs (86.7%), gears (95.0%), and connectors (100%), as well as moderately-high success rates (i.e., full insertions) across the same objects (76.7%, 92.5%, and 85%). The **Pegs and Holes** assembly **Insert** policy also successfully generalized to NEMA connectors, which can be considered extensions of peg-in-hole.

Engagement failures were almost exclusively due to slip between the gripper and object; we hypothesize that a high-force gripper (e.g., Robotiq) would fully resolve this issue. Full-insertion failures were almost exclusively due to the wedging phenomenon, a longstanding topic of research [71].

Informally, when the robot was intentionally perturbed by a human during the **Gears and Gearshafts** assembly **Insert** policy, the policy exhibited recovery behavior. In addition, the policy exhibited search behavior on the surface of the socket tray, exploring the vicinity of the observed goal (Figure 7).

E. Pick, Place, and Insert (PPI) Demonstration

This experiment demonstrated the ability of the robot to execute end-to-end assembly; the **Insert** policies required robustness to error accumulated from **Pick** and **Place**.

Experimental Setup: 6 pegs, 3 gears, 2 NEMA connectors, and their corresponding sockets were initialized with the same conditions as the **Pick** experiment. The goal was for the robot to bring all parts into their assembled configurations.

Key Results: The system demonstrated even higher success rates than during the **Insert** experiment: 80% and 88.3% success/engagement rates for peg insertion, 97.5% and 100% success/engagement rates for gears, and 100% success/engagement rates for connectors. The higher success rates suggest that the randomization and noise ranges during the **Insert** experiment may have been particularly adverse.

To our knowledge, **IndustReal** is the first system to demonstrate RL-based sim-to-real transfer for the end-to-end assembly task (i.e., detection, grasping, part transport, and insertion) without any policy adaptation phase in the real world.

[4]To our knowledge, only two other sim-to-real efforts have examined perturbations of this magnitude [11, 79]. Both manipulated much larger pegs.

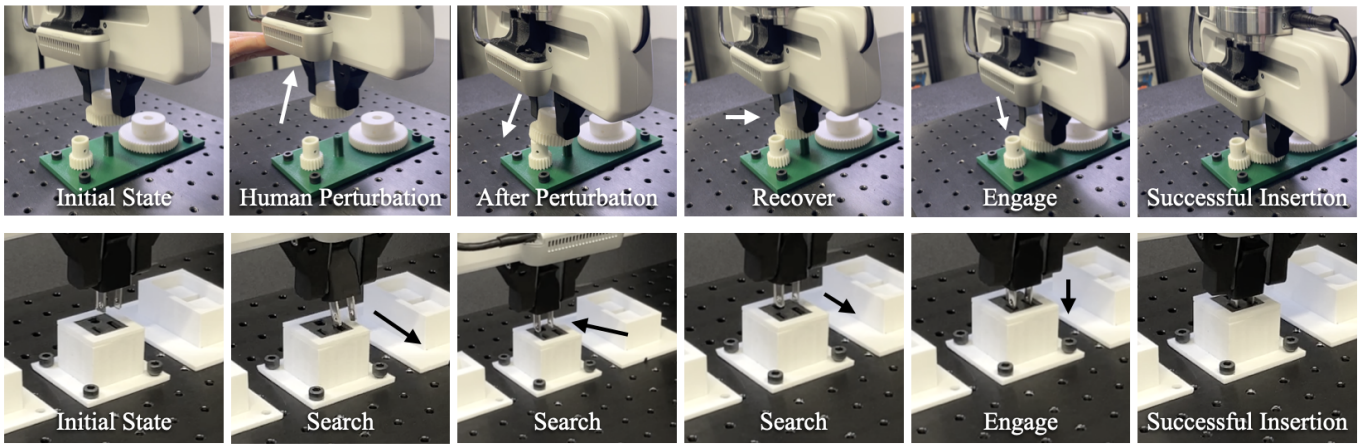


Fig. 7: **Snapshots of real-world experiments.** Top row: recovery behavior exhibited by the robot after human perturbation during gear insertion. Bottom row: search behavior exhibited during 3-prong NEMA connector insertion.

Asset	Pick	Insert		Pick-Place-Insert	
	Success	Success	Engage	Success	Engage
Round peg 8 mm	19/20	7/10	7/10	7/10	7/10
Round peg 12 mm	19/20	7/10	9/10	7/10	7/10
Round peg 16 mm	20/20	8/10	10/10	8/10	10/10
Rectangular peg 8 mm	20/20	8/10	9/10	10/10	10/10
Rectangular peg 12 mm	20/20	8/10	8/10	8/10	9/10
Rectangular peg 16 mm	20/20	8/10	9/10	8/10	10/10
NEMA 2-prong	-	10/10	10/10	10/10	10/10
NEMA 3-prong	-	7/10	10/10	10/10	10/10
Small gear	-	8/10	9/10	10/10	10/10
Medium gear	-	9/10	9/10	9/10	10/10
Large gear	-	10/10	10/10	10/10	10/10
Multi-gear assembly	-	10/10	10/10	10/10	10/10
Total #	158/160	100/120	110/120	107/120	113/120
Total (%)	**98.75%**	**83.33%**	**91.67%**	**89.16%**	**94.17%**

TABLE III: **Real-world experimental results for Pick, Insert, and PPI.**

VII. IndustRealKit and IndustRealLib

We strongly encourage fellow researchers to reproduce our results and use our platform to investigate their own questions. As follows, we will open-source our two most critical pieces of hardware and software, **IndustRealKit** and **IndustRealLib**.

IndustRealKit contains 3D-printable CAD models for all parts we designed, as well as a list of all parts we purchased from external vendors (Figure S11). The CAD models include 20 parts: 6 peg holders, 6 peg sockets (i.e., extruded holes), 3 gears, 1 gear base (with gearshafts), and 4 NEMA connectors and receptacle holders. The purchasing list includes 17 parts: 6 metal pegs (from the NIST benchmark), 4 NEMA connectors and receptacles, 1 optical breadboard, and fasteners.

IndustRealLib is a lightweight library containing code for policy deployment and training. Specifically, we provide scripts to allow users to quickly deploy policies from Isaac Gym [43] onto a Franka robot. The scripts include a base class that implements our policy-level controllers and sends/receives actions and observations from *FrankaPy*; task-specific classes that interpret actions from the policy, compute observations, and set targets; and a script that instantiates the classes, loads corresponding policies, and executes them. For the **Peg and Hole** assemblies, we also provide weights for the **Reach**, **Pick**, **Place**, and **Insert** policies. We have thus far used **IndustRealLib** on two different Franka robots in two different cities. Finally, we provide code for training our RL policies, including implementations of **SAPU**, **SDF-Based Reward**, and **SBC**. This code also includes a carefully-reviewed Franka model and simulation parameters validated during this work.

VIII. Limitations & Future Work

Our work has limitations, which lend themselves naturally to future research directions. First, like other efforts on sim-to-real transfer for assembly tasks, we have primarily investigated tasks inspired by the NIST benchmark [25]. However, recent work in graphics and robotics has provided a large number of simulation-compatible assets for assembly (e.g., [62]), potentially enabling RL policies that can generalize across widely different categories. Second, our primary failure cases on the real system were due to slip of the object in the gripper and wedging of plugs in their corresponding sockets. We believe that these cases can be resolved by providing the agent with simulated visuotactile readings during training [55, 74, 68] and using corresponding sensors on the real-world system [77, 28], as well as more accurately simulating friction [4]. Third, we do not explore passive mechanical compliance as a means for facilitating policy learning [46]; we believe that optimizing the policy, controller, and passive dynamics simultaneously can significantly help improve task performance. Fourth, our sim-to-real framework currently relies on a high-accuracy simulator and our proposed training- and deployment-time algorithms. However, for some tasks of even higher complexity (e.g., assembly of elastic cables onto pulleys), the simulator may neither be fundamentally accurate nor efficient enough to smoothly train and deploy policies to the real world. We envision the construction of a tight feedback loop from real-world deployments (i.e., a sim-to-real-to-sim loop) as a potentially compelling training strategy [1, 33].

IX. Conclusions

In this paper, we have presented **IndustReal**, a set of algorithms, systems, and tools to solve benchmark assembly problems in simulation and transfer policies to the real world. The utility of our simulation-based algorithms (**SAPU**, **SDF-Based Dense Reward**, and **SBC**) and real-world algorithm (**PLAI**) has been demonstrated through careful experiments in simulation and the real world. We provide the first simulation results for a series of benchmark tasks proposed in [48], and most critically, we demonstrate what is, to our knowledge, the first real-world system for RL-based sim-to-real on the end-to-end assembly task with no policy adaptation phase. Finally, in the hope of full reproducibility, we provide hardware and software for others in the community to replicate our results.

Acknowledgments

The authors thank Michael Noseworthy for advice throughout the project, Bowen Wen and Ajay Mandlekar for advice on perception, Eric Heiden and Balakumar Sundaralingam for advice on writing Warp kernels for SAPU, Lucas Manuelli for help with the initial implementation of IndustRealLib and advice on dynamics and control, Karl Van Wyk and Nathan Ratliff for feedback on PLAI, Viktor Makoviychuk for advice on domain randomization and asymmetric actor-critic, Gavriel State and Kelly Guo for providing support with Isaac Gym, Philipp Reist and Tobias Widmer for providing support with PhysX, and Sandeep Desai and Kenneth Maclean for providing support with hardware setup and 3D printing.

Contributions

Bingjie T., Yashraj N., Fabio R. developed SAPU.

Bingjie T., Yashraj N., Dieter F. developed SDF reward.

Bingjie T., Yashraj N. developed SBC.

Michael L., Yashraj N. developed PLAI.

Michael L., Yashraj N., Iretiayo A., Bingjie T. developed IndustRealLib.

Yashraj N., Michael L. developed IndustRealKit.

Bingjie T., Yashraj N., Ankur H. developed the perception module.

Bingjie T., Michael L. ran experimental evaluations.

Yashraj N., Bingjie T., Michael L. wrote the paper.

Yashraj N., Bingjie T., Michael L., Fabio R., Ankur H., Gaurav S. revised the paper.

Bingjie T., Yashraj N., Michael L. created the video.

Yashraj N., Dieter F., Fabio R., Gaurav S., Ankur H., Iretiayo A. advised the project.

References

[1] Saminda Abeyruwan, Laura Graesser, David B D'Ambrosio, Avi Singh, Anish Shankar, Alex Bewley, and Pannag R Sanketi. i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops. *arXiv preprint arXiv:2207.06572*, 2022.

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] 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. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[4] Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH Courses*. 2022.

[5] 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. *International Journal of Robotics Research*, 2020.

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

[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. *Applied Sciences*, 2020.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

[9] Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand dexterous manipulation from depth. *arXiv preprint arXiv:2211.11744*, 2022.

[10] Yunuo Chen, Minchen Li, Wenlong Lu, Chuyuan Fu, and Chenfanfu Jiang. Midas: A multi-joint robotics simulator with intersection-free frictional contact. *arXiv preprint arXiv:2210.00130*, 2022.

[11] Todor Davchev, Kevin Sebastian Luck, Michael Burke, Franziska Meier, Stefan Schaal, and Subramanian Ramamoorthy. Residual learning from demonstration: Adapting DMPs for contact-rich manipulation. *IEEE Robotics and Automation Letters*, 2021.

[12] Samuel Hunt Drake. *Using compliance in lieu of sensory feedback for automatic assembly*. PhD thesis, Massachusetts Institute of Technology, 1978.

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

[14] Zachary Ferguson, Minchen Li, Teseo Schneider, Fran-

cisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M Kaufman, and Daniele Panozzo. Intersection-free rigid body dynamics. *ACM Transactions on Graphics*, 2021.

[15] Letian Fu, Huang Huang, Lars Berscheid, Hui Li, Ken Goldberg, and Sachin Chitta. Safely learning visuotactile feedback policies in real for industrial insertion. *arXiv preprint arXiv:2210.01340*, 2022.

[16] Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca. Dynamic identification of the Franka Emika Panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 2019.

[17] Sami Haddadin, Sven Parusel, Lars Johannsmeier, Saskia Golz, Simon Gabl, Florian Walch, Mohamadreza Sabaghian, Christoph Jähne, Lukas Hausperger, and Simon Haddadin. The Franka Emika robot: A reference platform for robotics research and education. *IEEE Robotics and Automation Magazine*, 2022.

[18] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. DeXtreme: Transfer of agile in-hand manipulation from simulation to reality. *arXiv preprint arXiv:2210.13702*, 2022.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[20] 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.

[21] Zhimin Hou, Jiajun Fei, Yuelin Deng, and Jing Xu. Data-efficient hierarchical reinforcement learning for robotic assembly control applications. *IEEE Transactions on Industrial Electronics*, 2021.

[22] Shouren Huang, Kenichi Murakami, Yuji Yamakawa, Taku Senoo, and Masatoshi Ishikawa. Fast peg-and-hole alignment using visual compliance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[23] 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.

[24] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[25] 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 Robotics and Automation Letters*, 2020.

[26] Kenneth Kimble, Justin Albrecht, Megan Zimmerman, and Joe Falco. Performance measures to benchmark the grasping, manipulation, and assembly of deformable objects typical to manufacturing applications. *Frontiers in Robotics and AI*, 2022.

[27] Shir Kozlovsky, Elad Newman, and Miriam Zacksenhouse. Reinforcement learning of impedance policies for peg-in-hole tasks: Role of asymmetric matrices. *IEEE Robotics and Automation Letters*, 2022.

[28] Mike Lambeta, Po-Wei Chou, Stephen Tian, Brian Yang, Benjamin Maloon, Victoria Rose Most, Dave Stroud, Raymond Santos, Ahmad Byagowi, Gregg Kammerer, Dinesh Jayaraman, and Roberto Calandra. Digit: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation. *IEEE Robotics and Automation Letters*, 2020.

[29] Lei Lan, Danny M Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. Affine body dynamics: Fast, stable & intersection-free simulation of stiff materials. *ACM Transactions on Graphics*, 2022.

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

[31] Michelle A Lee, Carlos Florensa, Jonathan Tremblay, Nathan Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[32] 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 Transactions on Robotics*, 2020.

[33] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. In *International Conference on Robotics and Automation (ICRA)*, 2022.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[35] Tomas Lozano-Pérez, Matthew T Mason, and Russell H Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 1984.

[36] 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.

[37] 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 preprint arXiv:2103.11512*, 2021.

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

[39] 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.

[40] Miles Macklin. Warp: A High-performance Python Framework for GPU Simulation and Graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[41] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. Local optimization for robust signed distance field collision. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2020.

[42] Denys Makoviichuk and Viktor Makoviychuk. rl-games: A High-performance Framework for Reinforcement Learning. https://github.com/Denys88/rl_games, May 2022.

[43] 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.

[44] Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

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

[46] Andrew S Morgan, Bowen Wen, Junchi Liang, Abdeslam Boularias, Aaron M Dollar, and Kostas Bekris. Vision-driven compliant manipulation for reliable, high-precision assembly tasks. *arXiv preprint arXiv:2106.14070*, 2021.

[47] Fabio Muratore, Michael Gienger, and Jan Peters. Assessing transferability from simulation to reality for reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[48] Yashraj Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, et al. Factory: Fast contact for robotic assembly. In *Robotics: Science and Systems*, 2022.

[49] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[50] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[51] 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.

[52] 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.

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[54] 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.

[55] Zilin Si and Wenzhen Yuan. Taxim: An example-based simulation model for GelSight tactile sensors. *IEEE Robotics and Automation Letters*, 2022.

[56] 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.

[57] Oren Spector and Dotan Di Castro. InsertionNet: A scalable solution for insertion. *IEEE Robotics and Automation Letters*, 2021.

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

[59] Oren Spector, Vladimir Tchuiev, and Dotan Di Castro. InsertionNet 2.0: Minimal contact multi-step insertion using multimodal multiview sensory input. *arXiv preprint arXiv:2203.01153*, 2022.

[60] Te Tang, Hsien-Chung Lin, Yu Zhao, Wenjie Chen, and Masayoshi Tomizuka. Autonomous alignment of peg and hole by force/torque measurement for robotic assembly. In *IEEE International Conference on Automation Science and Engineering (CASE)*, 2016.

[61] 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.

[62] Yunsheng Tian, Jie Xu, Yichen Li, Jieliang Luo, Shinjiro Sueda, Hui Li, Karl DD Willis, and Wojciech Matusik. Assemble them all: Physics-based planning for generalizable assembly by disassembly. *ACM Transactions on Graphics (TOG)*, 2022.

[63] Roger Y Tsai and Reimar K Lenz. A new technique for

fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 1989.

[64] 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.

[65] 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.

[66] 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. *Advanced Robotics*, 2020.

[67] 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.

[68] 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. *IEEE Robotics and Automation Letters*, 2022.

[69] Bowen Wen, Wenzhao Lian, Kostas Bekris, and Stefan Schaal. You only demonstrate once: Category-level manipulation from single visual demonstration. *arXiv preprint arXiv:2201.12716*, 2022.

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

[71] Daniel E Whitney et al. Quasi-static assembly of compliantly supported rigid parts. *Journal of Dynamic Systems, Measurement, and Control*, pages 65–77, 1982.

[72] 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.

[73] Yanchun Xia, Yuehong Yin, and Zhaoneng Chen. Dynamic analysis for peg-in-hole assembly with contact deformation. *The International Journal of Advanced Manufacturing Technology*, 2006.

[74] Jie Xu, Sangwoon Kim, Tao Chen, Alberto Rodriguez Garcia, Pulkit Agrawal, Wojciech Matusik, and Shinjiro Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2022.

[75] Jing Xu, Zhimin Hou, Zhi Liu, and Hong Qiao. Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies. *arXiv preprint arXiv:1904.05240*, 2019.

[76] Jaemin Yoon, Minji Lee, Dongwon Son, and Dongjun Lee. Fast and accurate data-driven simulation framework for contact-intensive tight-tolerance robotic assembly tasks. *arXiv preprint arXiv:2202.13098*, 2022.

[77] Wenzhen Yuan, Siyuan Dong, and Edward H Adelson. GelSight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors*, 2017.

[78] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-Interface and FrankaPy. *arXiv preprint arXiv:2011.02398*, 2020.

[79] 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.

[80] Tony Z Zhao, Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Nicolas Heess, Jon Scholz, Stefan Schaal, and Sergey Levine. Offline meta-reinforcement learning for industrial insertion. In *International Conference on Robotics and Automation (ICRA)*, 2022.

APPENDIX

A. Related Work

Here we review classical and learning-based approaches to robotic assembly and briefly comment on simulation.

1) Classical Approaches: Assembly has been an open challenge in robotics for decades [70, 45]. Many analytical methods have been proposed to solve assembly tasks, particularly for the canonical problem of peg-in-hole insertion; these methods have typically utilized geometry, dynamics, mechanical design, and sensing as fundamental tools. Drake [12] proposed remote center compliance (RCC) as a means to mitigate reliance on visual or force sensing during assembly. Whitney et al. [71] described the effects of part geometry, gripper and support stiffness, and friction on contact forces and adverse outcomes (e.g., jamming and wedging). Lozano-Pérez et al. [35] proposed compliant motion planning strategies for assembly. Xia et al. [73] derived a compliant contact model and used the model to avoid jamming and wedging. Huang et al. [22] addressed initial part misalignment using vision, and Tang et al. [60] addressed this challenge via force/torque sensing. The preceding principles and methods are the prevailing means of addressing the annual NIST Assembly Task Board challenge, the established benchmark in robotic assembly [25, 66]. Nevertheless, such methods can be highly sensitive to errors in modeling, sensing, and state estimation; perturbations of adapters, fixtures, and calibration; and the introduction of unseen or more complex assets.

2) Learning-Based Approaches: In the past few years, learning-based approaches to assembly have gained popularity in the robotics research community, with many of the efforts focused on RL. Earlier works have typically explored model-based algorithms, such as guided policy search (GPS) [61] and iterative linear-quadratic-Gaussian control (iLQG) [36]. Despite the sample efficiency of these algorithms, nonlinear and discontinuous contact dynamics have made them challenging to leverage for contact-rich manipulation tasks [58].

A number of recent works have used model-free, off-policy RL algorithms, including classical Q-learning [23], deep-Q networks (DQN) [79], soft actor-critic (SAC) [7], probabilistic embeddings for actor-critic RL (PEARL) [52], hierarchical RL [21], and most popularly, deep deterministic policy gradients (DDPG) [6, 39, 37, 65]. These algorithms are also sample efficient, but can have unfavorable convergence properties.

A smaller number of research efforts have used model-free, on-policy algorithms, including trust region policy optimization (TRPO) [32], proximal policy optimization (PPO) [20, 56, 67, 48], and asynchronous advantage actor-critic (A3C) [54]. These algorithms have favorable convergence properties and are easy to tune; however, they are highly sample inefficient and can require long training times.

Other recent works have used on-policy or off-policy RL algorithms that can leverage demonstrations (e.g., human demonstrations or reference trajectories and controllers), such as residual learning from demonstration (rLfD), minimum-jerk trajectories, or impedance controllers, [11, 27, 24], interleaved Riemannian Motion Policies (RMP) and SAC [31], guided DDPG [13], DDPG from demonstration (DDPGfD) [37, 38, 64], offline meta-RL with advantage-weighted actor-critic (AWAC) [80], and inverse RL [72]. For efforts using human demonstrations, substantial engineering infrastructure and collection time can be required; demonstrations can be suboptimal; and successful demonstrations can be difficult to reliably obtain during high-precision tasks.

Several learning-based, non-RL approaches have also been proposed. These approaches include using human-initialized self-supervised learning to learn a policy or residual policy with multimodal inputs [57, 59, 15], as well as learning from videos of human demonstrations using category-level visual representations and 6-DOF tracking [69].

The preceding efforts define the state-of-the-art in learning-based approaches for assembly. Several have demonstrated high success rates and repeatability, shown robustness to small perturbations of initial part poses, and/or shown some degree of generalization across parts; one has even outperformed a solution from professional integrators [37]. However, most successful efforts have required human initializations, demonstrations, or on-policy corrections. Furthermore, purely real-world approaches are inherently difficult to parallelize; may require long training to achieve appreciable robustness (e.g., $\sim$50 hours in [37]); typically require manual resets; can be impractical for time-consuming, expensive, delicate, or dangerous tasks (e.g., construction); and do not fully leverage the substantial amount of virtual data available for industrial settings (e.g., nearly every existing industrial part originates from a CAD model that can be rendered or simulated).

3) Simulation: To our knowledge, the state-of-the-art in accurate and efficient contact-rich simulation is captured in [48, 41, 10, 29, 62, 76]. Among these, [48, 10] specifically address simulation for robotics tasks, whereas [48] integrates these capabilities within a widely-used robotics simulator [43]. Thus, we leverage [48] as our simulation platform.

B. Real-World System

1) Communications: A schematic of our communications framework is shown in Figure S8. The input to the communications pipeline is a trained RL policy (specifically, a checkpoint file) from Isaac Gym, which is provided to IndustRealLib. The output of the communications pipeline is a set of torque commands communicated to the robot.

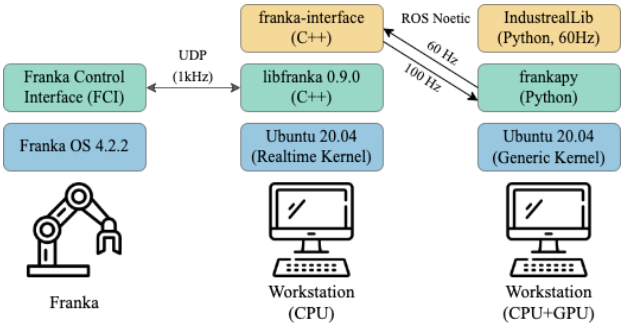
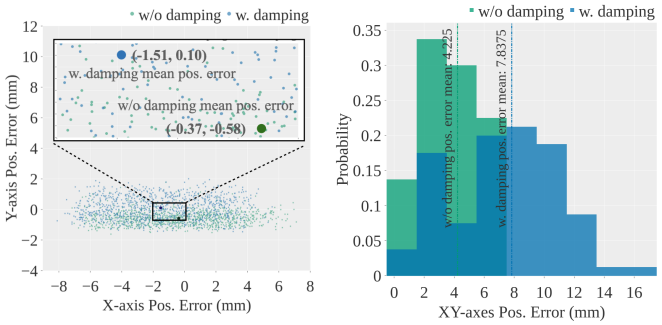


Fig. S8: Schematic of communications framework.



(a) Simulation (1000 trials) (b) Real world (160 trials)

Fig. S9: Evaluation of **Place** policy in simulation and the real world, with/without damping during simulation-based training.

2) Camera Calibration: The goal of intrinsic camera calibration for an RGB camera is to determine the relationship between the location of a 3D point in space and its location in the image. We rely on the intrinsic camera parameters provided by Intel through the *librealsense* and *pyrealsense2* libraries.

The goal of extrinsic camera calibration is to determine the 6-DOF pose T_{cam}^{robot} of the camera with respect to the robot frame. To compute extrinsic camera parameters, we first place a 6-inch AprilTag (52h13) onto the work surface and command the robot via the *frankx* library to random 6-DOF poses in the robot workspace, biased towards having the tag in view, but avoiding direct overhead views due to pose ambiguity. At each viewpoint, we detect the tag and compute the 6-DOF pose T_{tag}^{cam} of the tag with respect to the camera frame via the *pupil-apriltags* library, and we simultaneously query the 6-DOF pose T_{ee}^{robot} of the end-effector in the robot base frame. We collect approximately 30 such samples and then use the Tsai-Lenz method [63] from the *OpenCV* library to compute the 6-DOF pose T_{cam}^{ee} of the camera with respect to the end-effector. The 6-DOF pose T_{cam}^{robot} of the camera with respect to the robot base frame can then simply be computed as $T_{ee}^{robot}T_{cam}^{ee}$.

We validated the extrinsic parameters by comparing our estimated T_{cam}^{ee} against the corresponding transformation measured in a CAD assembly containing part models of the RealSense camera, camera mount, and end-effector.

3) Object Detection: The goal of our object detection module is to determine the object identities, 2D bounding boxes, and segmentation masks of each of our parts in the RGB image. To perform object detection, we used the implementation of the well-established Mask R-CNN [19] network architecture available in the *torchvision* library, which was pretrained on the Microsoft COCO dataset [34].

As the COCO dataset does not contain industrial assets, initial tests on our parts resulted in failure. Thus, we fine-tuned the pretrained model on real-world images of our assets. Specifically, we used the RealSense to capture 10-30 overhead images of each part randomly placed on our work surface, as well as 10 images of the work surface itself. We used Adobe Photoshop to automatically remove the backgrounds from the part images and extracted bounding boxes from the results.

We then divided the images into three different sets: 1) background, round pegs, rectangular pegs, round holes, and rectangular holes, 2) background, small gear, medium-sized gear, and large gear, and 3) background, NEMA 1-15 plug, NEMA 1-15 socket, USB-C plug, and USB-C socket.

For each set, we generated an augmented collection of images that consisted of each non-background element with random translations, rotations, and scaling. The elements were overlaid upon randomly-selected background images, and the composites were subject to color jitter using the *kornia* library.

Each augmented set of images was then used to train a Mask R-CNN model in *pytorch*. The categories within the set were added to the pretrained model, and the model was fine-tuned on images from the set to minimize losses over object identities, bounding boxes, and segmentation masks. Each model was trained for 50 epochs with 4000 training images, requiring $\approx$ 7.5 hours on a single GPU, at which point precision and recall scores were typically above 85%.

When using our trained models at test time, we performed additional data augmentation consisting exclusively of color jitter on captured images. The augmented images were used as input to our detection model. For each object in the image, the image with the highest object-identification confidence score was used to extract the object's identity, bounding box, and segmentation mask. This test-time augmentation improved our robustness to lighting variation and the presence of distractors. The yaw angle of each object was extracted by computing a minimum-area rectangle on the bounding box with *OpenCV* and calculating the angle of the box with the horizontal.

4) Workspace Mapping: For each detected object, we computed the centroid of its bounding box in image space. As mentioned earlier, the primary goal of our perception pipeline is to estimate the 2D poses (x, y, θ) of the parts on the workbench in the robot frame; thus, we needed to convert the location of the centroid (as well as the yaw angle determined earlier) from image space to 3D space.

In order to perform this transformation, we placed a 3-inch AprilTag (53h13) at an arbitrary location in the field of view and computed the pose T_{tag}^{cam} of the tag in the camera frame. With additional knowledge of the size of the tag and the pixel resolution of the camera, distances in image space were mapped to distances in the camera frame. (Strictly speaking, this relation holds only within the plane containing the AprilTag, with decreasing accuracy farther away due to perspective transformations.) The pose T_{tag}^{cam}, the location of the centroid of a particular part, and the distance mapping were used to convert the centroid and yaw angle from image space to the camera frame. Finally, the location of the centroid and the yaw angle were converted from the camera frame to the robot frame using the transformation T_{cam}^{robot} computed earlier.

We note that our workspace mapping process was susceptible to error induced by yaw rotations of the camera with respect to the robot frame, which resulted in systematic bias of our perceived locations relative to their actual locations. Thus, we performed a final one-time calibration step, during which we executed the Place experiment (i.e., laser tests) near the four corners of the workspace, estimated the offsets relative to

the centers of the targets, averaged the offsets, and subtracted the average from all subsequently-computed 3D locations.

5) Constructing Targets: After execution of the perception pipeline, the robot receives corresponding (x, y, $z = h_n$, $\alpha = \theta$, $\beta = 0.0$, $\gamma = 0.0$) as end-effector targets. The nominal height h_n at which to pick or place each part is specified in advance by the human; in our experience, for industrial-style parts, this specification is trivial (e.g., 25-50% from the top of the part) and requires little to no iteration.

6) Experimental Setup: Our real-world experimental setup consists of a Franka Emika Panda arm with a wrist-mounted RGB-D camera, as well as an optical breadboard upon which mechanical parts are placed. The robot, camera, optical breadboard, and parts are shown in Figure S11 and Figure S10.

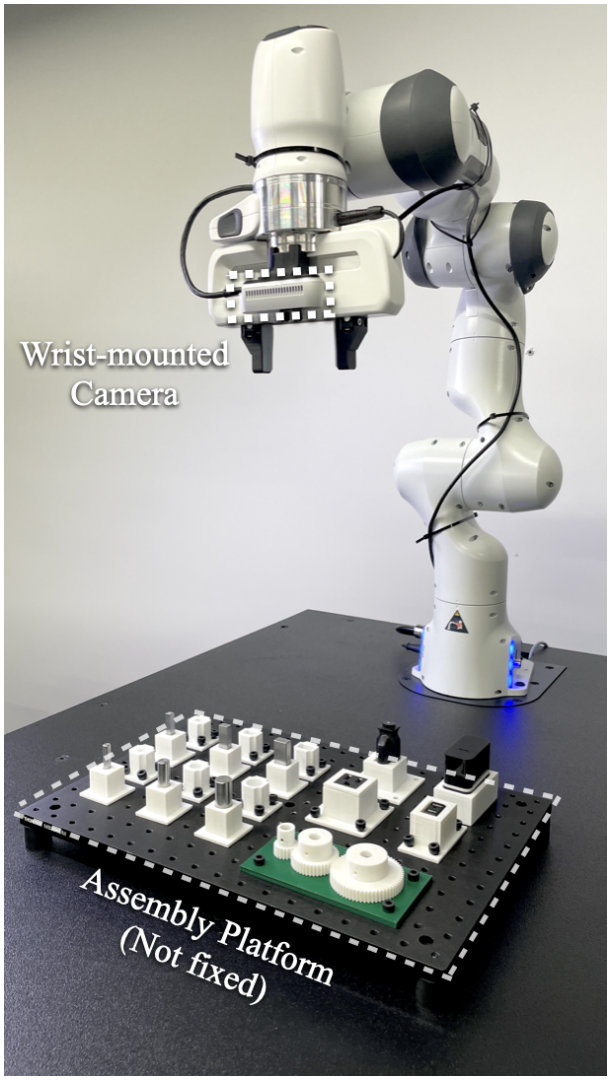


Fig. S10: Real-world experimental setup.

7) Summary: We have thus described our full perception pipeline. Camera calibration was performed once before beginning all experiments. The detection and workspace mapping process (aside from the one-time calibration step) were performed at the beginning of each trial that required detections; in total, these per-trial steps took approximately 10 seconds to execute on our non-realtime system.

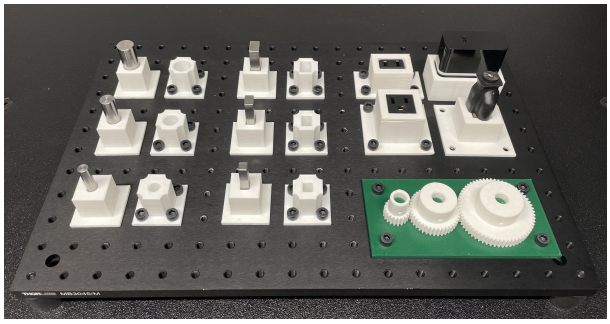


Fig. S11: A subset of parts from IndustRealKit.

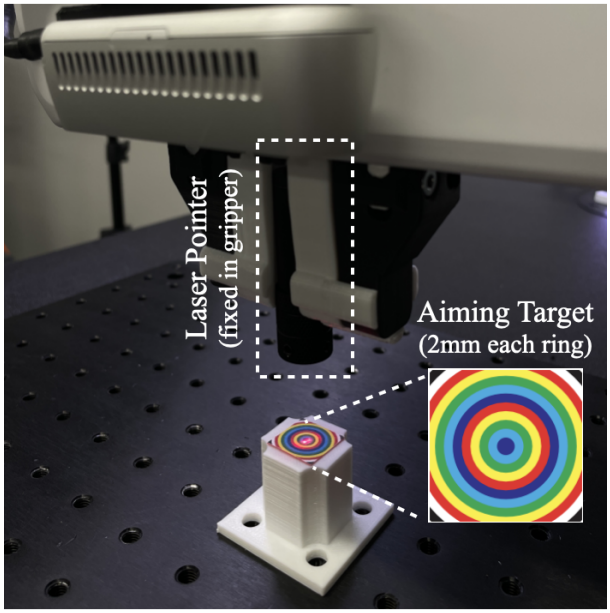


Fig. S12: Experimental setup for evaluating **Place** policy.

C. Policy-Level Action Integrator

1) Standard PLAI: Consider an RL policy that generates relative pose actions; in other words, the action composed with the current pose produces the desired state. In practice, the ability to achieve these desired states depends on passive and active system dynamics; having a large discrepancy between the environment used for training and deployment can lead to poor policy performance, especially when the system uses a more dynamic controller (e.g., a low stiffness task-space impedance controller). In this context, we briefly compare the behavior of **PLAI** with a lower-level proportional (P) controller, against a standard proportional-integral (**PI**) controller.[5]

The general form of a discrete-time P controller is given by

$$F[n] = k_P e[n] \tag{5}$$

[5]The comparison would also hold for PLAI with a lower-level PD controller, against a PID controller; we omit derivative terms for simplicity.

where F is the control effort (e.g., force or torque), k_P is the proportional gain, and e is an error signal.

We define the error signal as the difference between the desired state and the current state:

$$e[n] = x^d[n] - x[n] \tag{6}$$

where x is the state and superscript d denotes *desired*.

For **PLAI**, we apply actions to the previous desired state:

$$x^d[n] = \Pi(o[n]) + x^d[n-1] \tag{7}$$

In other words, we interpret actions as the difference between the current desired state and the previous desired state:

$$\Pi(o[n]) = x^d[n] - x^d[n-1] \tag{8}$$

We can re-write the error signal as

$$e[n] = \Pi(o[n]) + x^d[n-1] - x[n] \tag{9}$$

Next, we can roll out the first few timesteps of F:

$$\begin{aligned}
F[1] &= k_P\left(\Pi(o[1]) + x^d[0] - x[1]\right) \\
&= k_P\left(\Pi(o[1]) + \Pi(o[0]) + x^d[0] - x[1]\right) \\
F[2] &= k_P\left(\Pi(o[2]) + x^d[1] - x[2]\right) \\
&= k_P\left(\Pi(o[2]) + \Pi(o[1]) + \Pi(o[0]) + x^d[0] - x[2]\right) \\
F[3] &= k_P\left(\Pi(o[3]) + x^d[2] - x[3]\right) \\
&= k_P\left(\Pi(o[3]) + \Pi(o[2]) + ... + \Pi(o[0]) + x^d[0] - x[3]\right) \\
F[N] &= k_P(\Pi(o[N]) + \Pi(o[N-1]) + ... \\
&\quad + \Pi(o[0]) + x^d[0] - x[N])
\end{aligned}$$

More generally,

$$F[N] = k_P\left(\sum_{k=0}^{N}(\Pi(o[k])) + x^d[0] - x[N]\right) \tag{10}$$

In words, the sum of the policy outputs determines the control setpoint, which is tracked by the P controller. The summation term closely resembles an integral term in a **PI** controller:

$$F[N] = k_P e[N] + k_I \sum_{k=0}^{N}(e[k]) \tag{11}$$

$$= k_P(x^d[N] - x[N]) + k_I \sum_{k=0}^{N}(\Pi(o[k])) \tag{12}$$

The main difference is that the integral term in **PLAI** is used as a control setpoint, whereas in **PI**, it is directly converted into control effort. More extensively,

1) **PLAI** integrates the policy actions to generate a setpoint, which is used by the low-level impedance controller to attract the real state towards the desired state. If the system is disturbed from its current state (e.g., the robot is pushed away), the setpoint will not change instantaneously. Instead, the force vector will pull the system towards the same setpoint.
2) **PI** integrates the policy actions (in this case, equal to the control error) to generate a corrective force vector. If the system is disturbed from its current state, the accumulated error will be applied to an unintended state and may become a disturbance to the policy.

2) PLAI for 6-DOF Pose: The **PLAI** derivation above applies directly to control of Cartesian position; executing **PLAI** on Cartesian orientation is very similar, as addition and subtraction operators can be replaced by rotation matrix operations. Specifically, addition can be replaced with

$$R^O_{e_{n+1}} = R^O_{e_n} R^{e_n}_{e_{n+1}} \tag{13}$$

and subtraction can be replaced with

$$R^O_{e_n} = R^O_{e_{n+1}} [R^{e_n}_{e_{n+1}}]^T \tag{14}$$

where R^O_B is the rotation of a frame B relative to the frame O, and $R^{e_n}_{e_{n+1}}$ is an incremental rotation of the end-effector at time step $n+1$ relative to time step n.

3) Leaky PLAI: After the **PLAI** update (Equation 3), we can simply rewrite s^d_{t+1} as

$$s^d_{t+1} = s_t \oplus (s^d_{t+1} \ominus s_t) \tag{15}$$

and update the desired state as

$$s^d_{t+1} \leftarrow s_t \oplus \min\left((s^d_{t+1} \ominus s_t), \epsilon\right) \tag{16}$$

where ϵ is a threshold transformation.

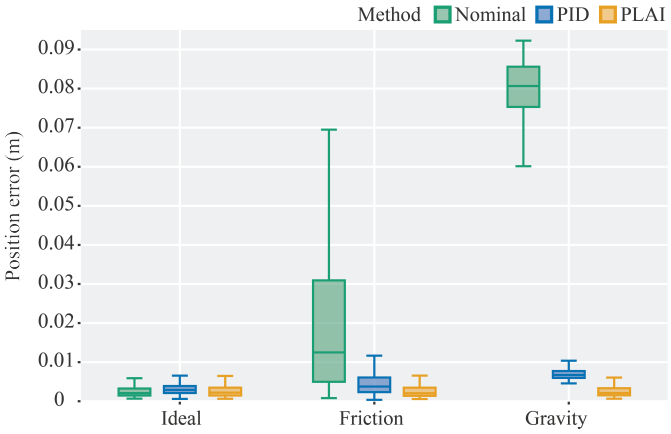


Fig. S13: Evaluation of **PLAI** in simulation. **PLAI** is compared to **Nominal** and **PID** under three environmental conditions. *Ideal* indicates no perturbations.

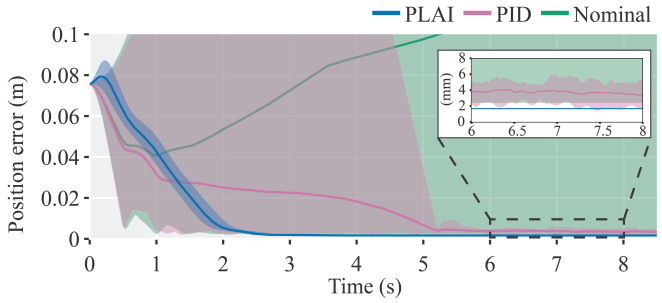


Fig. S14: Evaluation of **PLAI** in simulation, illustrating time-series behavior. **PLAI** is compared to **Nominal** and **PID** over 20 trials under a randomized gravitational disturbance (i.e., gravitational acceleration $g \sim U[0.1, 2.0] m/s^2$). Position error is measured with respect to final target. Inset view shows that steady-state error is lowest for **PLAI** (1.6 mm), followed by **PID** (3.5 mm); **Nominal** is frequently unable to converge.

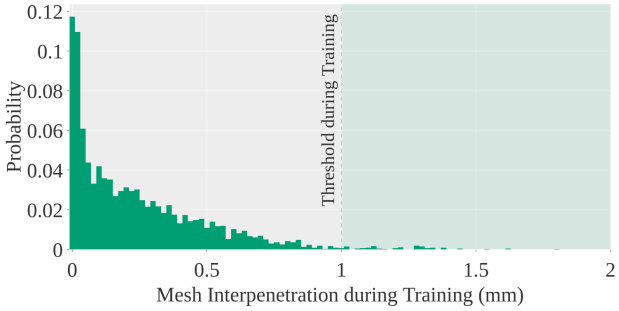


Fig. S15: Histogram of mesh interpenetrations during typical training episode.

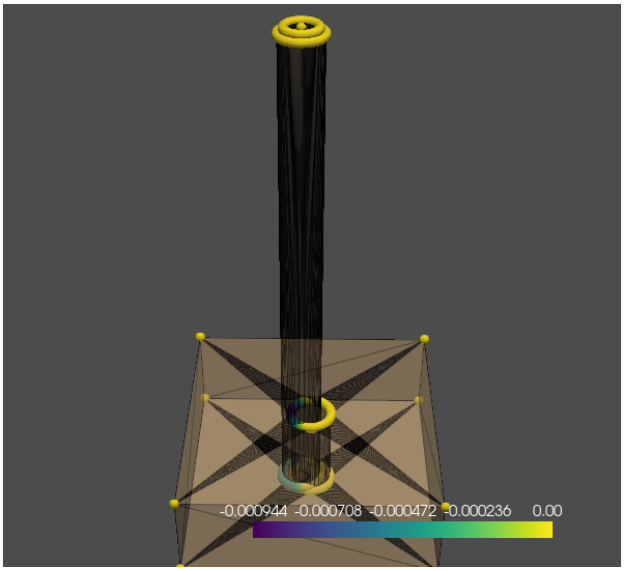


Fig. S16: Visualization of a transient interpenetration event between peg and hole assets with low-quality meshes. Yellow spheres denote mesh vertices. Colorbar is in m.

D. Additional Simulation Parameters & Results

1) MDP formulation & Parameters in Simulation: Given our choice of RL, we can formulate the problem as a Markov decision process (MDP) with state space $\mathcal{S}$, observation space $\mathcal{O}$, action space $\mathcal{A}$, state transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, initial state distribution ρ_0, reward function $r : \mathcal{S} \rightarrow \mathbb{R}$, horizon length T, and discount factor $\gamma \in (0, 1]$. The objective is to learn a policy $\pi : \mathcal{O} \rightarrow \mathbb{P}(\mathcal{A})$ that maximizes the expected sum of discounted rewards $\mathbb{E}_\pi[\Sigma_{t=0}^{T-1} \gamma^t r(s_t)]$.

We do not assume that state is fully observable in either simulation or the real world (specifically, $\mathcal{O} \subsetneq \mathcal{S}$). In the real world, the Franka's joint velocities, end-effector velocities, joint torques, and end-effector forces exhibit appreciable noise in free space. In addition, perceptual error leads to noise in object pose estimates, which in turn introduce noise into target poses. Furthermore, without tactile sensing or a 6-DOF tracker, we do not observe the state of objects within the gripper.

Table V describes the observation spaces, Table VI describes the reward formulations, and Table IV describes the task success criteria (i.e., the condition for a terminal reward) for policy training in simulation. In addition, Table VII describes the randomization range used for training.

2) Simulation-Aware Policy Update: Figure S15 shows the distribution of mesh interpenetration distances during a typical training episode. Figure S16 shows an example of mesh interpenetration between a peg and a hole.

3) Joint Evaluation in Simulation: Table X evaluates how the **Insert** policies perform under different randomization and observation noise conditions in simulation.

4) Additional Real Parameters & Results: Table VIII describes physics and control frequencies for simulation and the real world, Table IX describes real-world deployment parameters for the **Reach** policy, and Table III describes real-world deployment parameters for the remaining policies.

Task	Success Criterion	Condition
Pick	Successful lift	$h_{\text{obj}} > 3 * h_{\text{obj}} + h_{\text{table}}$
Place	Close placement	$\|\|\mathbf{k}^{\text{curr}} - \mathbf{k}^{\text{goal}}\|\|^2 < \epsilon_k$
Insert	Peg inserted in hole	$\Delta h < \epsilon_h$ & $\|\|\mathbf{k}^{\text{plug}} - \mathbf{k}^{\text{socket}}\|\|^2 < \epsilon_k$
Gear	Gear inserted on shaft	$\Delta h < \epsilon_h$ & $\|\|\mathbf{k}^{\text{gear}} - \mathbf{k}^{\text{shaft}}\|\|^2 < \epsilon_k$

TABLE IV: Success criterion for each policy. Symbols h_{obj} and h_{table} denote heights of the object and table, Δh denotes height from the hole/shaft base to the peg/gear base, $\mathbf{k}$ denotes keypoint positions, $\epsilon_k = 10\ cm$ is the keypoint distance error threshold, and $\epsilon_h = 3\ mm$ is the height error threshold.

Input	Dimensionality	Pick		Place		Insert (Pegs)		Insert (Gears)	
		Actor	Critic	Actor	Critic	Actor	Critic	Actor	Critic
Arm joint angles	7	✓	✓	✓	✓	✓	✓	✓	✓
Fingertip pose	3 (position) + 4 (quaternion)	✓	✓	✓	✓	✓	✓	✓	✓
Object (peg/gear) pose	3 (position) + 4 (quaternion)	✓	✓		✓		✓		✓
Target pose	3 (position) + 4 (quaternion)			✓	✓		✓		✓
Target pose with noise	3 (position) + 4 (quaternion)					✓		✓	
Relative target pose with noise	3 (position) + 4 (quaternion)					✓		✓	
Arm joint velocities	7		✓		✓		✓		✓
Fingertip linear velocity	3		✓		✓		✓		✓
Fingertip angular velocity	3		✓		✓		✓		✓
Object (peg/gear) linear velocity	3				✓		✓		✓
Object (peg/gear) angular velocity	3				✓		✓		✓
Target position observation noise	3						✓		✓
Relative target pose	3 (position) + 4 (quaternion)						✓		✓

TABLE V: Observations provided to the actor and critic for each policy. For **Pick**, *target pose* is the current plug pose; for **Place**, *target pose* is the target plug pose; and for **Insert (Pegs/Gears)**, *target pose* is a target end-effector pose.

Per-Timestep Reward	Formulation	Scale	Justification	**Reach**	**Pick**	**Place**	**Insert (Pegs)**	**Insert (Gears)**
Distance to goal	$-\|\|\mathbf{k}^{\text{curr}} - \mathbf{k}^{\text{goal}}\|\|^2$	1.0	Move closer to goal	✓	✓	✓		
SDF-based distance	$-\log(\Sigma_i^N \phi(x_i)/N)$	10.0	Align shapes				✓	✓
Scaling Factor on Return		**Range**						
Mesh interpenetration	$1 - \tanh(d/\epsilon_d)$	$[0, 1]$	Avoid interpenetration				✓	✓
Randomization range	$\zeta_{\text{curr}}/\zeta_{\max}$	$[0, 1]$	Adapt to randomization				✓	✓
Curriculum difficulty	$D_{\text{curr}}/D_{\max}$	$[0, 1]$	Adapt to difficulty				✓	✓
Bonus scale	$1/(\triangle h + 0.1)$	$[0, 1]$	Move closer to goal				✓	✓
Sparse Reward	**Condition**	**Value**						
Task success	See Table IV	10.0	Complete task		✓	✓	✓	✓
Close to target	$\|\|\mathbf{k}^{\text{curr}} - \mathbf{k}^{\text{goal}}\|\|^2 < \epsilon_k$	1.0	Move close to target		✓	✓		

TABLE VI: Rewards for each policy. Symbol $\mathbf{k}$ denotes keypoint positions, x_i denotes vertices on the plug, N is the number of vertices sampled from the plug mesh, $\phi(x_i)$ is the SDF value at a given vertex, d is the mean mesh interpenetration distance, ϵ_d is the interpenetration distance threshold, ζ is the magnitude of the randomization range, D is the stage of the curriculum, Δh is the height difference between current pose and goal pose when the episode is marked as successful, and ϵ_k is the keypoint distance error threshold. Note that the reward formulations for the **Reach**, **Pick**, and **Place** policies are simple; those for **Insert (Pegs)** and **Insert (Gears)** required more nuance.

Reach		**Pick & Place**		**Insert (Pegs)**		**Insert (Gears)**	
Parameter	Range	Parameter	Range	Parameter	Range	Parameter	Range
Hand X-axis	[0.3, 0.7] m	Hand X-axis	[0.3, 0.7] m	Plug to socket ΔXY	[-10, 10] mm	Gear to shaft ΔXY	[-10, 10] mm
Hand Y-axis	[-0.35, 0.35] m	Hand Y-axis	[-0.35, 0.35] m	Plug to socket ΔZ	[0, 20] mm	Gear to shaft ΔZ	[0, 20] mm
Hand Z-axis	[0.05, 0.35] m	Hand Z-axis	[0.05, 0.35] m	Socket X-axis	[0.4, 0.6] m	Shaft X-axis	[0.4, 0.6] m
Hand rotation	[-15, 15] deg	Hand rotation	[-15, 15] deg	Socket Y-axis	[-0.1, 0.1] m	Shaft Y-axis	[-0.1, 0.1] m
Target X-axis	[0.35, 0.65] m	Object X-axis	[0.35, 0.65] m	Socket Z-axis	[0.0, 0.05] m	Shaft Z-axis	[0.0, 0.05] m
Target Y-axis	[-0.15, 0.15] m	Object Y-axis	[-0.15, 0.15] m	Socket yaw	[-5, 5] deg	Shaft yaw	[-15, 15] deg
Target Z-axis	[0.0, 0.05] m	Object Z-axis	[0.0, 0.05] m	Observation noise	[-1, 1] mm	Observation noise	[-1, 1] mm

TABLE VII: Ranges for initial pose randomization and observation noise during training. Values were uniformly sampled.

	Physics	**Observations**		**Actions**	
		Low-Level	Policy	Low-Level	Policy
Simulation	120 Hz	60 Hz	60 Hz	60 Hz	60 Hz
Real-world	c/L	1000 Hz	60 Hz	1000 Hz	60 Hz

TABLE VIII: Physics and control frequencies for simulation and reality. Physics frequency in the real world is given by c/L, where c is the speed of sound and L is a characteristic length-scale; c can be approximated by the Newton-Laplace equation.

Deploy Method	**Parameter**	**Value**
All	Controller Gains	[1000, 1000, 1000, 50, 50, 50]
PLAI	Pos. Action Scale	[0.0005, 0.0005, 0.0005]
PLAI	Rot. Action Scale	[0.001, 0.001, 0.001]
Nominal	Pos. Action Scale	[0.01, 0.01, 0.01]
Nominal	Rot. Action Scale	[0.01, 0.01, 0.01]

TABLE IX: Reach policy evaluation parameters in real robot

Randomization	Insert (Pegs)				Insert (Gears)			
	Success (%)	Engage (%)	Pos. Err. (mm)	Rot. Err. (rad)	Success (%)	Engage (%)	Pos. Err. (mm)	Rot. Err. (rad)
Peg/Gear Pos. (mm)								
$[0, 0]$	92.40±2.30	98.40±1.67	2.83±0.46	0.075±0.0058	82.40±3.58	84.40±3.78	10.92±1.16	0.30±0.055
$[-5, 5]$	92.40±1.52	97.80±1.48	2.98±0.42	0.075±0.0083	78.40±3.97	81.80±3.27	12.32±2.14	0.26±0.042
$[-10, 10]$	88.60±2.41	96.60±0.023	3.80±0.00080	0.086±0.026	82.00±3.54	85.20±2.68	11.09±1.92	0.29±0.040
$[-15, 15]$	75.20±6.49	89.60±3.91	46.16±38.71	0.45±0.25	71.88±2.62	80.68±1.36	12.90±1.16	0.22±0.075
$[-20, 20]$	69.00±5.10	85.4±5.81	53.12±38.92	0.33± 0.059	71.60±1.33	80.88±0.85	12.66±1.24	0.22±0.016
Hole/Shaft Pos. (cm)								
$[0, 0]$	85.00±2.49	95.00±1.58	3.17±0.30	0.074±0.011	78.20±3.56	79.20±3.70	19.75±7.21	0.40±0.024
$[-5, 5]$	89.80±0.015	97.40±0.015	2.58±0.36	0.065±0.019	82.60±4.34	84.60±3.85	14.22±2.12	0.30±0.16
$[-10, 10]$	88.60±2.41	96.60±0.023	3.80±0.00080	0.086±0.026	82.00±3.54	85.20±2.68	11.09±1.92	0.29±0.040
$[-15, 15]$	78.60±2.07	86.20±1.92	7.10±0.44	0.085±0.0052	67.86±3.47	75.88±2.58	14.37±1.84	0.22±0.0088
$[-20, 20]$	69.40±4.16	78.00±2.55	10.38±1.14	0.11±0.0065	67.88±2.56	73.18±2.36	15.59±1.88	0.25±0.016
Observation Noise (mm)								
$[0, 0]$	87.00±2.55	93.60±1.52	4.84±.51	0.091±0.021	81.80±1.64	84.00±0.020	11.94±1.09	0.28±0.045
$[-1, 1]$	88.60±2.41	96.60±0.023	3.80±0.00080	0.086±0.026	82.00±3.54	85.20±2.68	11.09±1.92	0.29±0.040
$[-2, 2]$	86.80±2.28	97.00±2.35	3.89±0.56	0.085±0.011	79.40±2.70	82.20±4.09	11.40±1.49	0.27±0.044

TABLE X: Joint evaluation of Simulation-Based Policy Update, SDF-Based Reward, and Sampling-Based Curriculum. (A) **Pegs and Holes** assembly **Insert** policy. (B) **Gears and Gearshafts** assembly **Insert** policy. *Engage* denotes partial insertion. Policies were trained with moderate randomization (i.e., plug randomization of ± 10 mm, socket randomization of ± 10 cm, and observation noise of ± 1 mm); thus, the table evaluates in-distribution and out-of-distribution performance. Each test was executed on 5 seeds, with 1000 trials each.

	Reach	**Pick**	**Place**	**Insert (Pegs)**	**Insert (Gears)**
MLP network size (Actor)	[256, 128, 64]	[256, 128, 64]	[256, 128, 64]	[512, 256, 128]	[512, 256, 128]
MLP network size (Critic)	[512, 256, 128]	[512, 256, 128]	[512, 256, 128]	[256, 128, 64]	[256, 128, 64]
LSTM network size (Actor)	-	-	-	256	256
Horizon length (T)	128	128	128	256	256
Adam learning rate	1e-4	1e-4	1e-4	1e-3	1e-3
Discount factor (γ)	0.99	0.99	0.99	0.998	0.998
GAE parameter (λ)	0.95	0.95	0.95	0.95	0.95
Entropy coefficient	0.0	0.0	0.0	0.0	0.0
Critic coefficient	2	2	2	2	2
Minibatch size	2048	2048	2048	2048	2048
Minibatch epochs	8	8	8	8	8
Clipping parameter (ϵ)	0.2	0.2	0.2	0.2	0.2

TABLE XI: PPO parameters and network architectures used in each policy.

Task	**Asset**	**Parameter**	**Value**
Pick & Place	All	Controller Gains	[1000, 1000, 1000, 50, 50, 50]
Pick & Place	All	Pos. Action Scale	[0.002, 0.002, 0.0015]
Pick & Place	All	Rot. Action Scale	[0.004, 0.004, 0.004]
Insert	Pegs & Connectors	Controller Gains	[1000, 1000, 100, 50, 50, 50]
Insert	Pegs	Pos. Action Scale	[0.0006, 0.0006, 0.0004]
Insert	Connectors	Pos. Action Scale	[0.0004, 0.0004, 0.0004]
Insert	Pegs & Connectors	Rot. Action Scale	[0.001, 0.001, 0.001]
Insert	Pegs	Leaky PLAI Threshold	[0.05, 0.05, 0.03]
Insert	Connectors	Leaky PLAI Threshold	[0.04, 0.04, 0.05]
Insert	Gears	Controller Gains	[300, 300, 300, 50, 50, 50]
Insert	Gears	Pos. Action Scale	[0.0005, 0.0005, 0.0004]
Insert	Gears	Rot. Action Scale	[0.001, 0.001, 0.001]
Insert	Gears	Leaky PLAI Threshold	[0.05, 0.05, 0.05]
Insert	All	Observation Noise	XY-axes: [-2, 2] mm
Insert	All	Pos. Randomization	XY-axes: [-10, 10] mm
Insert	All	Height Randomization	[10, 20] mm above socket/shaft
Insert	All	Rot. Randomization	Yaw: [-5, 5] deg
Insert	All	Clearance	[0.5, 0.6] mm

TABLE XII: Real-world evaluation parameters for **Insert**. *Action scales* are scalars applied to position and rotation actions in order to bring robot execution speeds to within comfortable limits.