Specialist Assembly Policies over Diverse Geometries

Bingjie Tang[1], **Iretiayo Akinola**[2], **Jie Xu**[2], **Bowen Wen**[2], **Ankur Handa**[2], **Karl Van Wyk**[2],

Dieter Fox[2,3], **Gaurav S. Sukhatme**[1], **Fabio Ramos**[2,4], **Yashraj Narang**[2]

[1]University of Southern California, [2]NVIDIA Corporation,
[3]University of Washington, [4]University of Sydney

Abstract: Robotic assembly for high-mixture settings requires adaptivity to diverse parts and poses, which is an open challenge. In this work, we present a novel simulation-based approach for learning specialist (i.e., part-specific) policies. The trained specialist policies can individually solve 80 diverse assemblies with ≈80%+ success rates in simulation. We also demonstrate zero-shot sim-to-real transfer that achieves similar (or better) performance than simulation, including on perception-initialized assembly. For videos and additional details, please see our project website. [1]

Keywords: Assembly, Contact-rich manipulation, Sim-to-real transfer.

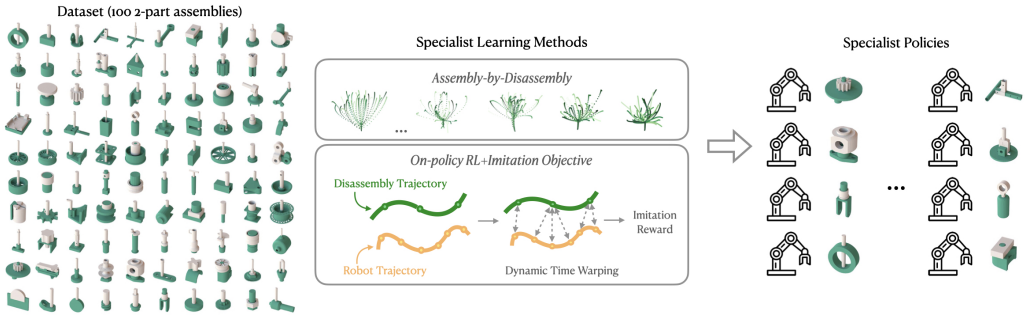

Figure 1: **Overview.** We present a dataset of 100 interpenetration-free assemblies that can be simulated in robotics simulators and assembled in the real world; specialist (i.e., part-specific) policies trained with a novel approach combining assembly-by-disassembly, RL with imitation, and dynamic time warping, which can solve 80 assemblies with ≈80%+ success rates.

1 Introduction

Most objects in home and industrial settings consist of multiple parts that must be assembled [1]. Human workers typically perform assembly; however, in certain industries (e.g., automotive), robotic assembly is prevalent. As industrial robots typically use stiff controllers and perform repetitive motions, robotic assembly requires highly-customized engineering of fixtures, tooling, and waypoints. Nevertheless, in high-mixture settings, *adaptive* assembly is required [2], in which robots must assemble parts with diverse geometries and poses. Adaptive assembly is non-trivial even for skilled human workers and is a major open challenge in robotics.

In this context, we present a novel approach combining 3 distinct algorithms for learning specialist policies: assembly-by-disassembly, reinforcement learning (RL) with an imitation objective, and dynamic time warping. These approaches are a synthesis of diverse algorithms from distinct fields,

[1]This workshop submission only partially covers our work **"AutoMate: Specialist and Generalist Assembly Policies over Diverse Geometries"** at *Robotics: Science and Systems 2024*, for the full version of the paper, please see our project website.

8th Conference on Robot Learning (CoRL 2024), Munich, Germany.

including manufacturing engineering, character animation, and time-series analysis. On 100 distinct assembly tasks, the learned specialist policies in simulation can individually solve ≈80 assemblies with 80%+ success rates over 500k trials. We also demonstrate a real-world system that can deploy our specialist policies in zero-shot with 86.5% success rates over 20 assemblies and 200 trials.

2 Related Works

There are few directly-comparable works for learning specialist policies for assembling a large number of diverse parts (here, 100). Recent efforts have focused on perception [3, 4, 5, 6] or planning [7, 8] without learning policies robust to disturbances and noise, or learning policies for a small number of assemblies (1-5), with just a few efforts attempting >10 assemblies [9, 10, 11]. Thus, we instead review works addressing challenges that we faced when learning specialist policies: 1) generating demonstrations for robotic assembly in simulation, 2) augmenting RL with demonstrations, and 3) selecting relevant demonstrations to use during learning.

For (1), the prevailing approach is assembly-by-disassembly (i.e., generating disassembly sequences/paths and reversing them for use in assembly), which was developed in the context of manufacturing engineering [12]. State-of-the-art tree search methods for this process are proposed in [7, 8]. Importantly, physical laws dictate that only sequences and paths can be reversed (rather than velocities and accelerations) [13].

For (2), there is a diverse set of effective approaches, including bootstrapping RL with behavior-cloned policies [14], adding demonstrations to the replay buffer for off-policy RL [15], augmenting the policy gradient for on-policy RL [14], learning a reward function from demonstrations [16] or human preferences [17], and explicitly including an imitation objective in the reward function for on-policy RL [18, 19], which was proposed in the character-animation literature.

Finally, for (3), the simplest approach is to select the demonstration closest to the initial position of the end effector. However, as we show, selecting the closest demonstration to the *current* position at each timestep produces more robust behavior, and selecting the closest demonstration to the *history* of positions (i.e., the end-effector path) is even more effective. Matching paths of varying lengths and discretizations is a well-known challenge; two state-of-the-art methods are dynamic time warping (DTW) [20] and signature transforms [21], mathematical techniques that were first applied to time-series analysis in speech recognition [22] and finance [23].

Our work combines the strengths of the preceding works by proposing a novel approach combining 3 algorithms: 1) assembly-by-disassembly, 2) RL with an imitation objective, and 3) trajectory matching via DTW. We select (2) for its simplicity and demonstrated effectiveness and uniquely formulate our imitation objective to imitate paths rather than states or state-action pairs. We select (3) based on subsequently-described evaluations. This combination enables effective training of specialist policies for ≈80% of our assemblies.

3 Learning Specialist Policies for Assembly

Our fundamental task is to use off-the-shelf, research-grade robot hardware (i.e., a collaborative robot arm with a parallel-jaw gripper, a low-cost RGB-D camera, and a second parallel-jaw gripper mounted to the work surface) to assemble a wide range of assemblies. Unlike most prior efforts, the assemblies consist of small parts with diverse geometries, the parts are initialized with appreciable 6-DOF pose randomization, no part-specific adapters or fixtures are leveraged, and no force-torque sensor is used. For a formal problem statement, see **Appendix A**.

For specialist policies, we find that RL alone is ineffective; thus, we guide RL with imitation learning. We face 3 challenges: 1) generating demonstrations for assembly, 2) augmenting RL with demonstrations, and 3) selecting demonstrations to use during learning. To address these challenges, we propose a novel approach combining assembly-by-disassembly, RL with an imitation

objective, and trajectory matching via dynamic time warping and signature transforms. We describe these building blocks in **Sections 3.1, 3.2, and 3.3**, respectively.

3.1 Assembly-by-Disassembly

When training specialists, our first challenge is to generate demonstrations for assembly. Collecting human demonstrations for assembly in simulation is challenging, requiring skilled operators and advanced teleoperation interfaces [24]. However, using motion planners is also difficult, as the kinematics of assembly are a narrow-passage problem [25]. Inspired by [12, 7], we instead generate demonstration paths for disassembly, which we reverse to generate paths for assembly.

Specifically, we first perform kinematics- and geometry-based grasp sampling based on [26]; for details, see **Figure S8** and **Appendix B**. For each assembly, we sample grasps along the surface of the plug, reject the samples if they violate kinematic or manipulability constraints, and repeat the process until generating 100 grasp candidates.

Next, we implement physics-based grasp evaluation. For each grasp, we randomize the pose of the assembly (**Table 2**), command the robot to execute the grasp using an impedance controller, and lift the plug to a random pose (**Figure 2**). If the plug remains in the gripper after the procedure, the grasp is successful. We repeat the process for 1000 trials and compute success rates for all 100 grasps; the highest-performing grasp for the given assembly is the one with the highest success rate.

For each assembly, now only using the highest-performing grasp, we repeat the evaluation procedure. For each successful trial, we consider the trajectory of the end effector as a disassembly demonstration D_i. In general, D_i consists of states $x = [x_i^1, x_i^2, ..., x_i^{N_i}]$, where N is the number of states; each state x_i^j can be defined as $[p_i^j; v_i^j; a_i^j]$, where p is position, v is velocity, and a is acceleration. We define a *reversed* disassembly demonstration D_i' as simply $x_i' = [x_i^{N_i}, x_i^{N_i-1}, ..., x_i^1]$, which can naively be used as an assembly demonstration. However, the corresponding velocities $v_i' = [v_i^{N_i}, v_i^{N_i-1}, ..., v_i^1]$ and accelerations $a_i' = [a_i^{N_i}, a_i^{N_i-1}, ..., a_i^1]$ are in general unrealistic or non-physical. Thus, for each successful trial, we record only the reversed disassembly *path* $p_i' = [p_i^{N_i}, p_i^{N_i-1}, ..., p_i^1]$. We repeat the procedure until collecting 100 successful reversed disassembly paths (**Figure 2D**), which we treat as assembly paths in **Sections 3.2 and 3.3**.

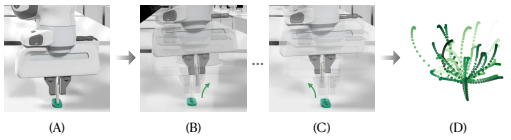

Figure 2: **Simulation-based generation of disassembly paths.** For each assembly, we generate disassembly paths by A) executing a grasp from a grasp optimization procedure, B) using an impedance controller to lift the plug from the socket and move to a randomized pose, and C) repeating the process for additional poses, until D) collecting 100 successful disassembly paths.

3.2 RL with Imitation Objective

When training specialists, our second challenge is to augment RL with demonstrations. Before we describe our augmentation approach, we briefly describe our baseline RL approach, which is a reimplementation of [27]; for RL formalism and extended descriptions, see **Appendix D**.

The training environments are initialized with a robot, a plug with randomized pose in the robot gripper, and a socket with randomized pose above a tabletop (**Table 2**). Ultimately, the robot must learn a policy that allows it to assemble the plug and socket while being robust to initial randomization, perceptual noise, and control error.

We formulate the robotic assembly problem as a Markov decision process (MDP), where the objective is to learn a policy that maximizes the expected sum of discounted rewards (i.e., solves the assembly problem in minimum time). We use proximal policy optimization (PPO) [28, 29] to learn the policy and an approximation of the value function (hyperparameters in **Table 3**).

Our observation space consists of joint angles, the current end-effector pose, and the end-effector goal pose; our input to the critic also includes joint velocities, end-effector linear/angular velocities, and the current plug pose (**Table 4**) [30]). To model real-world perceptual noise, we apply noise to all socket-pose observations (**Table 5**). Our action space consists of incremental (Δ) pose targets for a task-space impedance controller. Finally, our reward (without imitation) consists of terms that 1) penalize distance-to-goal, 2) penalize simulation error, and 3) reward task difficulty at each timestep, as well as a term that rewards task success (**Appendix D**).

Now we describe our augmentation approach. Inspired by [18, 19], we augment RL with demonstrations by directly adding an imitation reward to our reward formulation. Specifically, we define our per-timestep reward as follows:

$$R_t = \omega^B R_t^B + \omega^I R_t^I \tag{1}$$

where R_t^B is the baseline per-timestep reward, described above and in **Appendix D**; R_t^I is an imitation-based per-timestep reward that encourages the agent to mimic demonstrations; and ω^B and ω^I are weighting hyperparameters.[2]

Following [18], we define R_t^I as the maximum per-timestep reward over all demonstrations for the given assembly (i.e., the reversed disassembly paths from **Section 3.1**). Specifically,

$$R_t^I = \max_{i=1,...,M} R_t^{I_i} \tag{2}$$

where M is the number of demonstrations. Unlike [18, 19], we apply the augmentation approach to contact-rich manipulation rather than locomotion. We define $R_t^{I_i}$ in **Section 3.3**.

3.3 Trajectory Matching via Dynamic Time Warping and Signature Transforms

When training specialists, our third and final challenge is to select demonstrations to use during learning. Specifically, for a given assembly, we must define a reward $R_t^{I_j}$ that quantifies the instantaneous value of imitating any reversed disassembly path p'_i, after which we can use **Equation 2**.

Intuitively, we can define $R_t^{I_i}$ as the distance between the assembly path the robot has traversed during an RL episode, and the reversed disassembly path p'_i under consideration. However, our simulator [31] has a fixed Δt; thus, for a given path, the arc length between consecutive points is a function of the instantaneous velocity. In general, the path the robot has traversed has a disparate sequence of velocities compared to any reversed path p'_i, resulting in disparate spatial discretizations. We thus seek a distance metric between paths that is insensitive to speed or sampling rate.

We explore 2 methods for computing such a metric, dynamic time warping (**DTW**) and **signature transforms**. DTW is a dynamic programming algorithm for quantifying the difference between time series [22]. Given two sequences $a = [a^1, a^2, ..., a^P]$ and $b = [b^1, b^2, ..., b^Q]$, DTW matches each a^i to one or more b^j and vice versa. The matching process minimizes a cost $C(a, b)$ defined as the sum of Euclidean distances between each a^i and its match(es) from b; furthermore, the process satisfies constraints that 1) a^1 must match with at least b^1 (i.e., first points aligned), 2) a^P must match with at least b^Q (i.e., last points aligned), and 3) all matches must be monotonic (i.e., if a^i matches with b^j, then a^{i+1} cannot match with b^{j-1}, nor a^{i-1} with b^{j+1}). Ultimately, DTW returns the cost $C^*(a, b)$ of the optimal matches between a and b.

When we apply DTW, at each timestep t, we first extract the path $p_e(t, N)$ of the end effector over a window of length N=10 steps, $[p_e^{t-(N-1)}, p_e^{t-(N-2)}, ..., p_e^t]$. Then, for each reversed disassembly path p'_i, we find the closest subsequent points on p'_i from the first point $p_e^{t-(N-1)}$ and current point p_e^t on the windowed end-effector path, and we extract the segment of p'_i between those 2 closest points. We then use DTW to compute the minimum cost $C^*(p_e(t, N), p'_i)$ between the windowed end-effector path and the disassembly segment, and we set $R_t^{I_i} = 1 - \tanh C^*(p_e(t, N), p'_i)$. We repeat this procedure for each p'_i [32] and then compute R_t^I (**Equation 2**).

[2] We set ω^B and ω^I simply so that the baseline and imitation terms fall within the same order of magnitude.

On the other hand, **signature transforms** represent trajectories as a collection of path integrals called a *path signature* [33, 34, 21, 35], which can also quantify distances between paths. In our context, given a 3-dimensional path $p(t)_{a,b} = (x_1(t), x_2(t), x_3(t))_{a,b}$, where $x_1(t)$, $x_2(t)$, and $x_3(t)$ represent x, y, and z coordinates for $t \in [a, b]$, the path signature is a tensor of all possible path integrals between the coordinates. The *first level* of the path signature is

$$S_1(x_i(t))_{a,t} = \int_a^t dx_i(t) = x_i(t) - x_i(a) \tag{3}$$

where $i = 1, 2, 3$ (i.e., 3 total integrals), and the *second level* of the path signature is

$$S_2(x_i(t), x_j(t))_{a,t} = \int_a^t S_1(x_i(t))_{a,t} dx_j(t) \tag{4}$$

where $i = 1, 2, 3$ and $j = 1, 2, 3$ (i.e., 9 total path integrals). Further levels of the path signature can be derived in similar fashion. Finally, the full path signature is

$$S(p(t))_{a,b} = (1, S_1(x_i(t)_{a,b}), S_2(x_i(t), x_j(t))_{a,b}, ...) \tag{5}$$

where all indices iterate over $1, 2, 3$. The *signature transform* is simply the functional $T(p(t))_{a,b} : p(t)_{a,b} \rightarrow S((p(t))_{a,b}$ that takes a path as input and outputs the path signature. Path signatures inherit translation and reparameterization invariance from path integrals; as desired, these properties mitigate discretization sensitivity.

When we apply the signature transform, at each timestep t, we consider the full path $p_e(T)_{0,t}$ of the end effector from the beginning of the episode. Then, for each reversed disassembly path p'_i, we find the closest point on p'_i from the current point $p_e(t)$ on the end-effector path and extract the segment of p'_i between the start and the closest point. We then compute the path signatures $S(p_e(T))_{0,t}$ and $S(p'_i)$ of the end-effector path and disassembly path segment [36], respectively, and compute the cost $C(S(p_e(T))_{0,t}, S(p'_i))$ between signatures as

$$C\big(S(p_e(T))_{0,t}, S(p'_i)\big) = \|S(p_e(T))_{0,t} - S(p'_i)\|_2. \tag{6}$$

Finally, we set $R_t^{I_i} = 1 - \tanh C\big(S(p_e(T))_{0,t}, S(p'_i)\big)$.

4 Evaluations

We now present the results of our specialist policies, which are trained based on our combination of assembly-by-disassembly, RL with an imitation objective, and trajectory matching via dynamic time warping and signature transforms. We perform all evaluations in this section under the maximum bounds for initial-pose randomization (**Table 2**) and observation noise (**Table 5**) during training.

For specialist policies, our first evaluation question is, **which trajectory-matching approach is most effective?** We evaluate the following 4 test cases:

- **IndustReal** [27]: This is a state-of-the-art RL-only approach for simulation-based robotic assembly, described in **Section 3.2**. It does not use an imitation objective and thus illustrates results without trajectory matching.
- **State-based Matching**: This is a naive baseline for RL with an imitation objective. At each timestep, we calculate the distance from the end effector to every point of every disassembly path, and we compute our imitation reward based on the shortest distance.
- **Dynamic Time Warping (Ours)**: Imitation reward is based on the DTW distance.
- **Signature Transform (Ours)**: Imitation reward is based on a path-signature distance.

For each of the 100 assemblies, we train a specialist policy with the 4 preceding approaches for matching the current robot path with demonstrations. For each approach, we train 5 random seeds, select the best seed, and evaluate it 5 times over 1000 trials. We illustrate average results over all

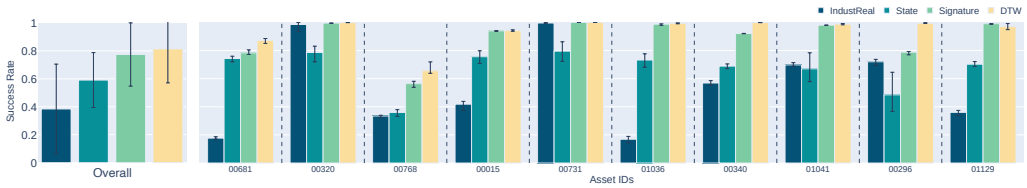

Figure 3: **Simulation-based evaluation of trajectory-matching approaches for learning specialist policies.** . *IndustReal* is a state-of-the-art matching-free approach. *State* selects the demonstration containing the closest point to the current robot state. *Signature* selects the demonstration with the minimum signature-transform distance from the robot trajectory. *DTW* selects the demonstration with the minimum dynamic-time-warping distance from the robot trajectory. The *Signature* and *DTW* approaches significantly outperform the others.

100 assemblies, as well as specific results for 10 sampled assemblies. **Figure 3** shows our results for for 10 sampled assemblies.
IndustReal has the lowest success rates over the 10 selected assemblies, indicating the importance of imitation. State-based matching provides a substantial improvement, but still does not result in high success rates. Dynamic time warping and signature transforms consistently have the highest success rates, with slightly better performance for the former. These results are also reflected over all 100 assemblies, with mean success rates of $38.45 \pm 32.16\%$ for IndustReal, $59.11 \pm 19.65\%$ for state-based matching, $81.50 \pm 24.42\%$ for DTW, and $77.46 \pm 22.61\%$ for signature transforms.

Our second evaluation question is, **how does our approach perform across all 100 assemblies?** Specifically, rather than selecting 10 assemblies or summarizing statistics, we tabulate results over all 100 assemblies. **Appendix E** shows our results.

Our approach shows consistent performance over a wide spectrum of the assemblies. Specifically, for 80 assemblies, our specialist policies has $\approx$80% (78%) success rates or higher, and for 55, it has 90% success rates or higher. We conclude that the proposed approach is a highly-effective strategy for training specialists over diverse geometries. The most challenging assemblies (i.e., bottom 20%) tend to have small-diameter plugs, sockets with small contact surfaces, or sockets with stair-like internal features.

5 Sim-to-Real Transfer

5.1 Real-World System Design

Our real-world system consists of a Franka robot with a parallel-jaw gripper, a wrist-mounted RealSense D435 camera, a Schunk EGK40 parallel-jaw gripper mounted to the work surface, and 3D-printed assemblies from our dataset (**Figure 4**). Our communications framework is closely modeled after [27]; however, our perception, grasping, and control procedures differ significantly.

For perception, we aim to accurately estimate plug and socket states while initializing them in a far less-constrained manner. We use a powerful segmentation tool [37], textureless CAD models of our parts, and a state-of-the-art pose estimator [38] to estimate the 6-DOF poses of each part from RGB-D images. For details, see **Appendix F**.

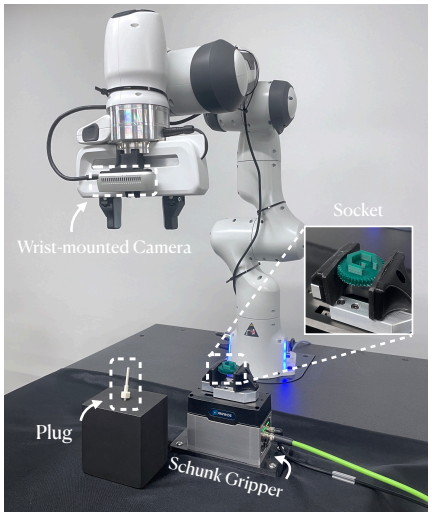

Figure 4: **Real-world experimental setup.** At the beginning of each experiment, a 3D-printed plug is haphazardly pressed into a foam block or placed in the robot gripper, and a 3D-printed socket is haphazardly placed in the Schunk gripper.

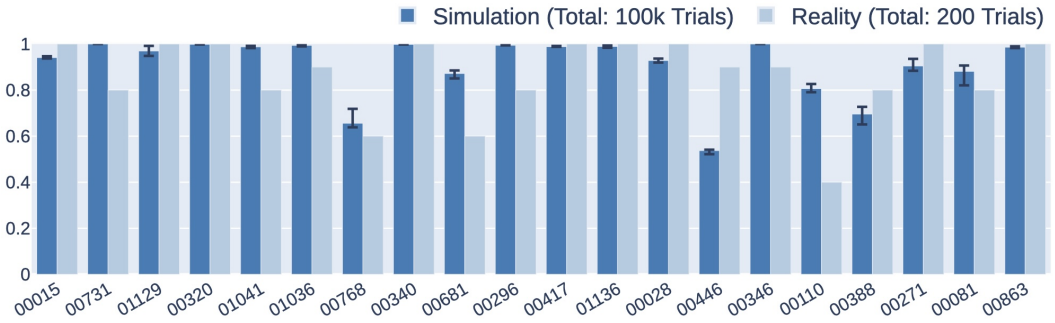

Figure 5: **Comparison of real-world specialist-policy success rates with simulated analogues.** We deploy our specialist policies over 20 assemblies and 200 trials (asset ID lookup in **Figure S12**). For specialists, our success rates in the real world are highly comparable to those in simulation, with a drop of only 4.15% on average.

For grasping, we aim to avoid manually specifying grasp poses. We use our grasp sampling and evaluation procedure described (**Section B**) to generate a high-performing grasp for each assembly, and we execute those grasps in the real world.

Finally, we aim to avoid any assembly-specific tuning of controller gains or action scales, and we restrict ourselves to a single set of parameters for all real-world trials.

5.2 Real-World Policy Evaluations

We now present the results of our specialist policies in the real world. For these trials, we place the robot in lead-through, manually grasp a plug, and guide it into the socket. We then programmatically lift the plug until free from contact; apply an xy perturbation of $\pm$ 10 mm, z perturbation of 15 $\pm$ 5 mm, and yaw perturbation of $\pm$ 5°; apply x, y, and z observation noise of $\pm$2 mm each; and deploy a policy.[3] For each of 20 assemblies shown in Figure 6, we deploy the corresponding specialist policy 10 times, for a total of 200 trials. **Figure 5** shows our results.

Figure 6: We show 20 sampled assemblies for real-world evaluation, with unique IDs listed for reference.

The mean success rate in the real world is 86.50 $\pm$ 16.52%, whereas the success rate in simulation is 90.65 $\pm$ 13.07%. Across assemblies, real-world success rates are within close range of simulation; in fact, for 11/20 assemblies, real-world is better. Such results likely indicate that our simulated training conditions (e.g., initial-pose randomization, observation noise) are sufficiently adverse to train robust and performant policies.

5.3 Real-World Perception-Initialized Evaluation

Finally, we present the results of our specialist policies as part of a perception-initialized assembly workflow. For these trials, we place the plug haphazardly on a foam block and place the socket haphazardly within the Schunk gripper. We capture an RGB-D image, estimate the poses of the parts, grasp the plug, transport it to the socket, and deploy a specialist assembly policy (**Figure 7**). This

[3]The manual grasping step is uncontrolled and likely contributes an additional 1-3 mm and 5-10 deg of perturbation.

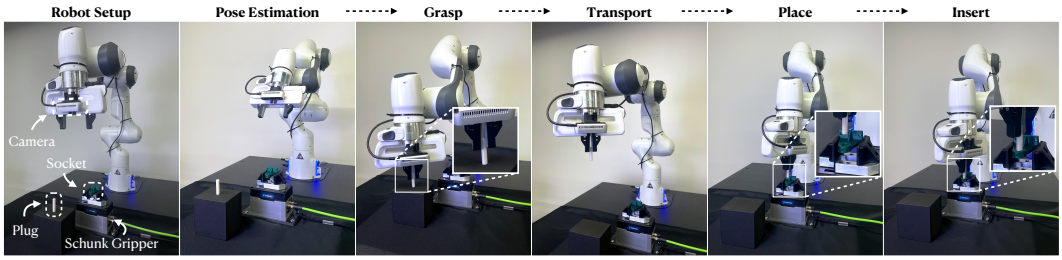

Figure 7: **Real-world perception-initialized assembly procedure.** We illustrate our procedure for performing perception-initialized assembly (i.e., from perception to insertion). Setup: We press a plug haphazardly onto a foam block, and we place a socket haphazardly within a Schunk gripper on a table. Pose Estimation: We estimate the pose of the plug or socket using our perception pipeline (**Section F**). Grasp: We grasp the plug using the output of our grasp optimizer (**Section B**). Transport: We transport the plug across the workspace. Place: We place the plug above the socket. Insert: We deploy a specialist or generalist policy (**Section 3**).

workflow presents a unique challenge, as initial part poses are even less constrained, and perception and/or control error accumulates at each stage, demanding increased policy robustness.

Our question is, **can our specialist policies solve the perception-initialized assembly task?** For 5 distinct assemblies, we deploy the corresponding specialists 10 times, for a total of 50 trials. **Table 1** shows our results.

For specialists, the mean success rate is 90.0%, which is within 4.0% of isolated policy evaluations from **Section 5.2**. These results indicate that 6-DOF pose estimation, grasp optimization, and our proposed methods for learning specialist policies can be effectively combined to achieve perception-initialized assembly. Qualitatively, higher success rates occur on assemblies with distinct visual features, as these correlate with more accurate pose estimates.

6 Conclusions

We present a novel approach combining 3 distinct algorithms for learning specialist policies: assembly-by-disassembly, reinforcement learning (RL) with an imitation objective, and dynamic time warping. To our knowledge, it provides the first simulation-based framework for learning specialist policies over a wide range of assemblies, as well as the first system to demonstrate zero-shot sim-to-real over such a range.

Asset ID	Specialist	
	Policy-Only	Perception-Init
00015	10/10	10/10
00296	8/10	8/10
00320	10/10	10/10
00340	10/10	9/10
00346	9/10	8/10
Total #	47/50	45/50
Total (%)	**94.0%**	**90.0%**

Table 1: **Real-world evaluations for perception-initialized assembly.** We provide success rates of our specialist policies as part of a perception-initialized assembly workflow. We also compare to isolated policy evaluations from **Section 5.2**

A key limitation of our work is that for 20 assemblies, our specialist policies achieve <80% success rates (**Figure S10**). Policy failures are typically caused by unstable grasps on irregular geometries or sudden slip between the contact surfaces of the plug and socket. We anticipate that simple hardware improvements (e.g., higher-force grippers) and algorithmic improvements (e.g., utilization of demonstrations where the robot recovers from slip) may resolve these failure cases.

Our work opens up exciting opportunities for future work, such as multi-part assemblies and trajectory-matching approaches that consist of SE(3) transforms. Through this line of work, we aim to gradually build towards a large-model paradigm for industrial robotics, while staying grounded in real-world deployment.

A Problem Description: Formal Problem Statement

Components:

- A robotic manipulator R with end effector EE
- A wrist-mounted RGB-D camera C
- A plug P with known geometry described by mesh M_P
- A socket S with known geometry described by mesh M_S

Definitions:

- The configuration of robot R is defined by joint angles $\Theta \in \mathbb{R}^7$.
- The pose of the end effector EE is denoted by $X_{EE} \in SE(3)$.
- The robot R is actuated by joint torques $T \in \mathbb{R}^7$.
- The camera C captures an initial image pair $\{I^0, D^0\}$ consisting of an RGB image I^0 and a depth image D^0.

Constraints:

- The relationship between the joint angles Θ and the end-effector pose X_{EE} is governed by a forward-kinematics model f:
$$X_{EE} = f(\Theta).$$
- The relationship between the joint torques T, joint angles Θ, and end-effector pose X_{EE} is governed by a control law g_ψ:
$$T = g_\psi\big(\Theta, X_{EE}, U(t)\big),$$
where ψ denotes constant parameters (e.g., control gains) and $U(t) \in \mathbb{R}^7 \cup SE(3)$ represents a control target in joint space ($\mathbb{R}^7$) or task space $\big(SE(3)\big)$ at the current timestep t. In our application, we use a task-space impedance controller (**Equation 8**).
- The initial poses of the end effector EE, plug P, and socket S are given by
$$\begin{aligned} X^0_{EE} &\sim \text{Uniform}(a^0_{EE}, b^0_{EE}), \\ X^0_P &\sim \text{Uniform}(a^0_P, b^0_P), \\ X^0_S &\sim \text{Uniform}(a^0_S, b^0_S), \end{aligned}$$
where a and b denote user-defined bounds of the initial-pose distribution for the corresponding variable, and initial poses X^0_P and X^0_S satisfy the constraint that the convex hulls of meshes M_P and M_S do not overlap. In our application, the bounds are specified in **Table 2**.
- The goal pose for the plug X^G_P satisfies the constraints that 1) the convex hulls of meshes M_P and M_S *do* overlap, 2) meshes M_P and M_S do not intersect, and 3) user-specified surfaces on plug P and socket S are in contact (typically, an inferior surface of P and superior surface of S).

Goal: The goal of the assembly task is to use joint angles Θ, images I^0 and D^0, and meshes M_P and M_S to generate the control inputs $U(t)$ that guide plug P from initial pose X^0_P to goal pose X^G_P.

B Methods: Grasp Optimization

The robot must first grasp the plug before assembling or disassembling the plug and socket, and the success of the assembly or disassembly process depends on the quality of the grasp. Thus, for each

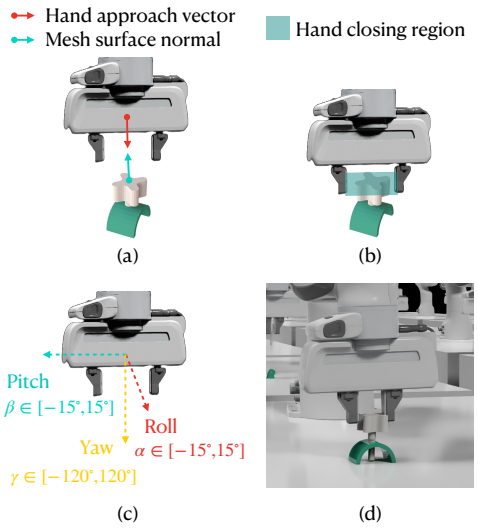

Figure S8: **Grasp sampling and evaluation procedure.** We apply a grasp sampling approach based on [26], and we develop a a physics-based evaluation procedure. A) We sample a surface normal on the socket mesh, align the gripper axis with the normal, sample a position along the normal, and translate the gripper to that position. B) We check if the plug lies within the gripper closing region. C) We check if the Euler angles of the gripper are within specified bounds. D) We use simulation to disassemble the plug from the socket and check whether the plug remains in the gripper fingers.

assembly, we perform an optimization procedure to determine a grasp that may lead to a high probability of success during assembly or disassembly. This high-performing grasp is then used when generating disassembly trajectories, training assembly policies, and deploying assembly policies in the real world for that assembly. Our grasp optimization procedure consists of the following steps:

1. **Grasp Sampling** (**Figure S8**): We apply a kinematics- and geometry-based grasp sampling approach based on [26]. For each assembly, we first initialize the plug and socket meshes in their assembled state. We instantiate a robot gripper mesh, randomly sample a surface normal on the plug mesh, and align the central axis of the gripper to be collinear with the surface normal. We then randomly sample a position along this normal and translate the gripper to that position. We define a *grasp sample* as the 6-DOF pose of the gripper in this state.

   We reject the grasp sample if 1) the robot hand intersects the plug or socket meshes, 2) the plug mesh does not intersect the gripper closing region (i.e., the prismatic volume contained between the fully-opened gripper fingers), or 3) the Euler angles of the gripper are outside of specified bounds ([-15, 15] deg for roll and pitch and [-120, 120] deg for yaw). The last of these criteria is designed to ensure that the Franka robot remains in a region of its workspace with high manipulability. We repeat this process until generating 100 grasp samples.

2. **Physics-Based Evaluation**: Although the preceding grasp sampling procedure provides kinematically-feasible grasps, these samples are not guaranteed to be stable during the contact-rich interactions experienced during assembly and disassembly. Thus, we develop a subsequent physics-based evaluation phase.

   For each assembly, we first randomize the pose of the socket over a wide range (**Table 2**), and we initialize the plug in its assembled state (i.e., inserted in the socket). For each of the 100 grasp samples, we execute the grasp on the plug. We use a task-space impedance controller [39] to lift the plug from the socket until the convex hull of the plug no longer intersects the convex hull of the socket, and we move the robot gripper to a pose in free space randomly sampled from specified bounds ([-0.05, 0.05] for X- and Y-position, [0,

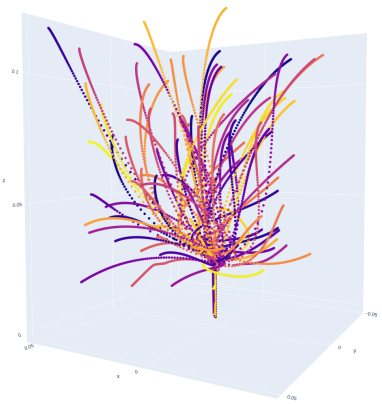

Figure S9: **Generated disassembly paths.** We generate disassembly paths via physics simulation and reverse the paths for use in assembly. Here we visualize 100 disassembly paths for an assembly with a deep socket.

0.05] for Z-position, and [-10, 10] deg for roll, pitch, and yaw). We check whether the grasp is successful (i.e., if the plug remains in the gripper fingers until the end of the procedure). We repeat this procedure 1000 times. Finally, we identify the grasp sample with the highest success rate and designate that sample as the highest-performing grasp for the given assembly. In total, we run 10 million trials (100 assemblies x 100 grasps per assembly x 1000 trials per grasp), but we distribute the evaluations over many parallel environments for efficiency.

Thus, the output of the grasp optimization procedure is a dictionary that maps each assembly to the highest-performing grasp for the corresponding plug. This grasp is inherently collision-free with respect to the socket in the assembled state, robust to large variations in robot configuration and plug/socket pose, and robust to contact-rich interactions.

C Methods: Disassembly Path Generation

Figure S9 shows a visualization of disassembly paths for a representative assembly.

D Methods: Reinforcement Learning

We formulate the robotic assembly problem as a Markov decision process (MDP), where the agent is a simulated robot, and the environment is a simulated environment containing the parts to be assembled. We define a state space $\mathcal{S}$, observation space $\mathcal{O}$, and action space $\mathcal{A}$. Our state-transition dynamics are defined by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, which is governed by the physical laws of rigid-body dynamics implemented in our simulator. We define a randomized initial state distribution ρ_0 (**Table 2**) and reward function $R : \mathcal{S} \rightarrow \mathbb{R}$ with discount factor $\gamma \in (0, 1]$. We constrain our agents to execute actions over episodes of length N timesteps, and we define shorter learning horizons of length T timesteps. Finally, our return G is defined as

$$G(T) = \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T-1}\gamma^t R(s_t)\right] \tag{7}$$

In other words, the return is the expected sum of discounted rewards over the horizon. The objective is to train a policy $\pi : \mathcal{O} \rightarrow \mathbb{P}(\mathcal{A})$ that maximizes the return.

To train policies, we use the proximal policy optimization (PPO) algorithm [29] due to its well-established performance over a wide range of simulation and sim-to-real problems [31], as well as its ease-of-use; we mitigate the low sample efficiency of PPO by using GPU-accelerated SDF-based contact simulation [40] and a GPU-accelerated PPO implementation [28]. We use PPO to learn a

	Parameter	Randomization Range
Socket	X-position (m)	[0.40, 0.60]
	Y-position (m)	[-0.10, 0.10]
	Z-position (m)	[0.16, 0.18]
	roll angle (deg)	[-5, 5]
	pitch angle (deg)	[-5, 5]
	yaw angle (deg)	[-5, 5]
Plug (rel. to socket)	X-position (mm)	[-10, 10]
	Y-position (mm)	[-10, 10]
	Z-position (mm)	[10, 20]
	roll angle (deg)	[-5, 5]
	pitch angle (deg)	[-5, 5]
	yaw angle (deg)	[-5, 5]
Plug (rel. to gripper)	X-position (mm)	[-1, 1]
	Y-position (mm)	[-1, 1]
	Z-position (mm)	[-1, 1]
	roll angle (deg)	[-5, 5]
	pitch angle (deg)	[-5, 5]
	yaw angle (deg)	[-5, 5]

Table 2: **Randomization ranges for initial object poses during training.** We list position and orientation ranges for the following randomization procedure: First, the socket position and orientation are randomized. Next, the plug position and orientation are randomized relative to the socket pose. Then, the robot is commanded to move its gripper near the plug pose. Finally, the plug position and orientation are randomized again relative to the gripper pose. Values are all sampled from uniform distributions.

Parameter	Specialist Training	Generalist Fine-tuning
MLP network size (actor)	[256, 128, 64]	[512, 256, 128, 64]
MLP network size (critic)	[256, 128, 64]	[256, 128, 64]
LSTM network size (actor)	256	256
Horizon length (T)	32	32
Adam learning rate	1e-4	1e-4
Discount factor (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Entropy coefficient	0.0	0.0
Critic coefficient	2	2
Minibatch size	8192	8192
Minibatch epochs	8	8
Clipping parameter (ϵ)	0.2	0.2

Table 3: **Network architectures and hyperparameters used with Proximal Policy Optimization (PPO).** We list our specialist and generalist policy network architectures for the actor and critic networks, as well as our most critical PPO hyperparameters. We use PPO to train specialist policies from scratch, as well as fine-tune generalist policies after distillation as described in **Section ??**.

stochastic policy π_θ (i.e., an actor) parameterized by a neural network with weights θ, as well as an approximation of the on-policy value function $V_\phi : \mathcal{S} \rightarrow \mathbb{R}$ (i.e., a critic) parameterized by a neural network with weights ϕ. At evaluation and deployment time, the actor is deterministic, and the critic is neglected. For network architectures and hyperparameters, see **Table 3**.

Our observation space provided to the actor consists of robot-arm joint angles [$\mathbb{R}^7$], the current pose of the end effector (i.e., the pose of the robot-gripper fingertips) [SE(3)], the goal pose of the end effector [SE(3)], and the pose of the end effector relative to the current pose [SE(3)] (**Table 4**). During training, the goal pose is simply the pose of the end effector when it is grasping the plug in its highest-performing grasp pose (**Appendix B**) while the plug is inserted into the socket. We avoid

including joint velocities, end-effector velocities, and joint torques in the observation space, as these measurements exhibit substantial noise in the real world and can impede sim-to-real transfer; we also avoid including plug pose, as measuring this quantity typically requires tactile sensing [41].

However, we adopt an asymmetric actor-critic strategy [30], where the states provided to the critic include privileged information that is not provided to the actor, as the critic is only used for training and is not deployed in the real world. Here, the states provided to the critic include joint velocities [$\mathbb{R}^7$], end-effector velocities [$\mathbb{R}^6$], and plug pose [SE(3)]. In addition, to capture real-world control error, perception error, and sensor noise, we apply uniformly-sampled noise to all observations of the position and orientation of the socket that are provided to the actor (**Table 5**), but do not apply noise to the corresponding states provided to the critic.

Input	Dimensions	Actor	Critic
Arm joint angles	7	✓	✓
Fingertip 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		✓
Plug pose	3 (position) + 4 (quaternion)		✓
Relative target pose	3 (position) + 4 (quaternion)		✓

Table 4: **Inputs to the actor and critic for specialist policies.** We list observations provided to the actor, as well as observations and states provided to the critic.

Parameter	Noise Range
Socket X-position	[-2, 2] mm
Socket Y-position	[-2, 2] mm
Socket Z-position	[-2, 2] mm
Socket roll angle	[-5, 5] deg
Socket pitch angle	[-5, 5] deg
Socket yaw angle	[-5, 5] deg

Table 5: **Noise ranges for observations of socket pose during training.** We list the ranges for position and orientation noise applied to observations of the socket pose, which is used to compute the goal pose of the end effector. Values were sampled from a uniform distributions.

Our action space consists of incremental pose targets [SE(3)], which represent the position and orientation difference between the current pose x_c and the target pose x_t; we choose incremental targets rather than absolute targets in order to select from a small, bounded spatial range. We pass these targets to a task-space impedance controller

$$\tau = J^T\left(k_p(x_t \ominus x_c) - k_d\dot{x}_c\right) \tag{8}$$

where $J \in \mathbb{R}^{6\times7}$ is the geometric Jacobian; $K_p \in \mathbb{R}^{6\times6}$ and $K_d \in \mathbb{R}^{6\times6}$ are diagonal matrices consisting of proportional and derivative gains, respectively; $\dot{x_c} \in \mathbb{R}^6$ is the velocity vector; and $x_t \ominus x_c$ computes the incremental pose target. We use a task-space impedance controller to generalize actions across robot configurations and avoid using inertial matrices, which have not been precisely measured for our robot manipulator. We superimpose a nullspace controller to softly constrain the robot to maintain a configuration with high manipulability, which can be compromised by elbow drift.

Finally, our baseline reward formulation (without imitation) is derived from [27] and is composed of terms that penalize distance-to-goal, penalize simulation error, reward task difficulty, and reward success. Specifically, the reward

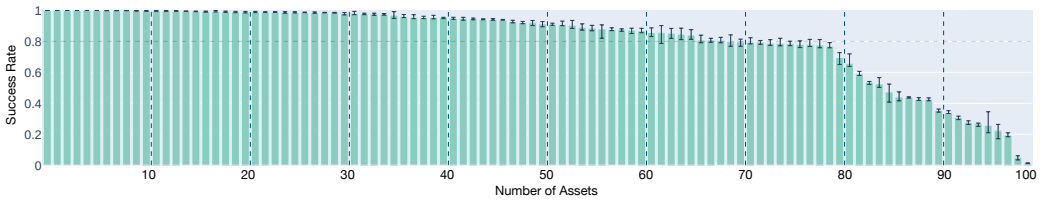

Figure S10: **Simulation-based evaluation of final training approach for specialist policies.** For each of the 100 assemblies, we train a specialist policy with the final AutoMate learning approach. For this approach, we train 5 random seeds, select the best seed, and evaluate it 5 times over 1000 trials. AutoMate maintains consistent performance across the majority of the assemblies and achieves the critical milestone of solving approximately 80% of the assemblies with 80% success rates or higher under substantial initial-pose randomization (**Table 2**) and observation noise (**Table 5**)

.

1. penalizes distance-to-goal through an SDF-based reward, which computes the distance between the current plug pose and the goal (i.e., assembled) plug pose through SDF queries, which are less sensitive to object symmetries than keypoint-based distance queries,

2. penalizes simulation error through a simulation-aware policy update (SAPU), which computes the maximum interpenetration distance at each timestep, weights the reward in inverse proportion to distance if it is less than a threshold, and does not update the reward otherwise,

3. rewards task difficulty through a sampling-based curriculum (SBC), which increases the lower bound (but not the upper bound) of the range of initial-pose randomization as the agent becomes more proficient at the task, and weights the return in proportion to task difficulty.

We defer precise descriptions to [27] and implementation details to [42]. Whereas [27] also rewarded success by providing a bonus at the end of every episode if a keypoint distance between the plug and its goal fell below a threshold on the final timestep, we instead reward success with a bonus at the end of every *horizon* if the *translational* distance between the plug and its goal falls below a threshold at *any* timestep.

Precisely, our return over each horizon is given as

$$G(T) = w_{SBC} \sum_{t=0}^{T-1} \left(\omega_{SAPU} (\omega_{SDF} R_{SDF} + \omega_I R_I) \right) + R_{succ} \tag{9}$$

where w_{SBC} is the weighting factor based on task difficulty, as determined by the SBC algorithm; ω_{SAPU} is the weighting factor based on simulation error, as determined by the SAPU algorithm; R_{SDF} is the distance-to-goal reward, as determined by the SDF-based reward; R_I is the imitation-based reward, as described in detail in the main text; ω_{SDF} and ω_I are hyperparameters to determine the relative importance of the distance-to-goal reward and the imitation-based reward; and R_{succ} is a success bonus applied at the end of each horizon. Parameters ω_{SDF} and ω_I are tuned simply so that R_{SDF} and R_I fall within the same order of magnitude.

E Results: Specialist Policies (Continued)

Figure S10 shows the results of our final training approach for specialist policies over all 100 assemblies.

F Methods: Perception

F.0.1 Camera Calibration and Tuning

We observe the environment with a single wrist-mounted Intel RealSense D435 RGB-D camera. It is a common sentiment among robotics researchers that off-the-shelf cameras may not be sufficient for high-precision tasks; moreover, it is common practice among the robot learning community to compensate for the weaknesses of such cameras primarily via data-driven strategies (e.g., increased data collection, data augmentation, etc.). However, we have found that diligent camera calibration and tuning can greatly improve the RGB image quality, point cloud quality, and performance in downstream perception modules (e.g., pose estimators), to a level sufficient for high-precision assembly of parts far smaller than the robot manipulator.

Specifically, we take the following steps:

- **Extrinsics calibration**: We calibrate our *absolute* extrinsics (i.e., the pose of our RGB camera in the robot frame) using the procedure described in [27], with a lightly-modified implementation of the corresponding code in [43]. The procedure consists of moving the end effector to randomized target poses, capturing an image of an AprilTag in each pose, computing the pose of the AprilTag in the camera frame from each image, and using the Tsai-Lenz algorithm [44] to compute the extrinsics matrix. We do not calibrate our *relative* intrinsics (i.e., the pose of our depth camera with respect to our RGB camera) and rely on the RealSense-provided matrix.
- **Intrinsics calibration**: We calibrate our intrinsics matrix using the procedure described in the RealSense whitepaper for on-chip self-calibration [45]. The procedure consists of capturing an image of a textured target and running the manufacturer-provided calibration function.
- **Camera settings**: We tune our camera settings using the suggestions provided in the RealSense whitepapers on tuning depth cameras [46] and depth image post-processing [47]. The most impactful settings were
  - RGB and depth camera exposure: We tune the exposure to maximize the quality of the color and depth map images on our assemblies, rather than using the default auto-exposure. Furthermore, we tune the RGB and depth camera exposures simultaneously (i.e., set them to the same value), rather than separately.
  - Laser power: We increase power from its default value to increase the density of the depth image.
  - Spatial hole-filling: We apply a hole-filling filter in post-processing to repair holes in the depth image.

Finally, to optimize the performance of our downstream pose estimator (described next), we maximized RGB camera resolution (in order to increase the number of pixels on small surfaces), captured images from angled (rather than overhead) view, and avoided direct lighting of the assemblies.

F.1 Pose Estimation

In simulation, we train RL policies from 6D poses of the parts rather than from RGB images or point clouds, which would substantially increase compute requirements. On the other hand, in the real world, we observe the environment using a single Intel RealSense D435 RGB-D camera mounted on the wrist of the robot. Thus, in order to deploy our simulation-trained policies in the real world, we may consider A) moving simulation towards reality (i.e., distilling the simulation-trained policies to use RGB-image and/or point-cloud inputs) or B) moving reality towards simulation (i.e., extracting 6D poses from real-world RGB images and/or point clouds). We choose option B for several reasons: 1) We can avoid a cross-modal distillation process, 2) Simulated and real-world RGB images have a substantial sim-to-real gap, 3) The quality of real-world point clouds is poor on

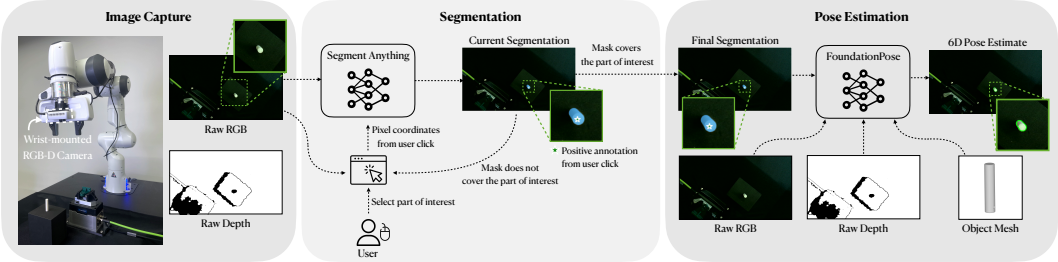

Figure S11: **Real-world perception pipeline.** Left: At the beginning of our pipeline, we use an Intel RealSense D435 RGB-D camera mounted on the wrist of the robot to capture an RGB image and depth image. Middle: We show the RGB image to the user, who clicks on the plug or socket of interest. We then pass the RGB image and pixel coordinates through a powerful segmentation tool [37] and compute a segmentation mask for the plug or socket. Right: We pass the RGB image, depth image, segmentation mask, and CAD model for the plug or socket into a state-of-the-art pose estimator [38] to estimate the 6-DOF pose of the part in the camera frame. We later use robot kinematics and camera extrinsics to convert the pose to the robot frame.

our small, mildly-reflective parts, and 4) The accuracy of pose estimators has improved dramatically in recent years [48, 49, 38].

We assume that each part of each assembly has a known CAD model (specifically, an OBJ file with no associated texture), which is typical in industrial assembly settings. We use a pose estimation pipeline that takes as input 1) an RGB-D image of a part in the real world, 2) the camera intrinsics matrix, 3) the camera extrinsics matrix, and 4) a CAD model of the part, and then predicts the 6D pose of the part. This pose can subsequently be passed as input to our RL policies.

Our pose estimation pipeline consists of the following steps:

1. **Image capture**: The RealSense camera is used to capture a 1280 x 720 RGB image and 1280 x 720 depth image of a part with a known CAD model.
2. **Part selection**: The RGB image is shown to the user. The user can left-click on the part to provide a positive annotation (i.e., a pixel that lies on the part of interest).
3. **Segmentation**: The RGB image, pixel location(s), and annotation(s) are passed to [37], which produces a high-accuracy segmentation mask for the part.
4. **Refinement (optional)**: If the mask does not span the part, the user can provide another positive annotation; conversely, if the mask includes background features, the user can right-click to provide a negative annotation (i.e., a pixel that does not correspond to the part). Segmentation is then executed with the additional annotations.
5. **Model-based estimation**: The RGB image, depth image, camera intrinsics, segmentation mask, and textureless CAD model are fed to [38], which regresses to the 6D object pose in the camera frame.
6. **Frame transformation**: The 6D object pose in the camera frame is combined with the camera extrinsics to compute the 6D object pose in the robot frame.

In total, the steps above take $\approx$20 seconds to execute.

Future work may focus on replacing the click-based interface for selecting parts with a language interface, using faster implementations of the segmentation model (e.g., [50, 51]), automatically retrieving the appropriate CAD model from a database, and optimizing [38] for faster performance.

Our real-world system consists of a Franka robot with a parallel-jaw gripper, a wrist-mounted RealSense D435 camera, a Schunk EGK40 parallel-jaw gripper mounted to the tabletop, and 3D-printed assemblies from our dataset (**Figure 4**). Our communications framework is closely modeled after [27]; however, our perception, grasping, and control procedures differ significantly.

Figure S12: **Assembly lookup chart.** For each assembly investigated in this work, we provide its unique assembly ID and a rendering. The assemblies are the same as those visualized in **Figure ??**. The asset IDs are referenced in figures throughout this paper.

For perception, we aim to estimate plug and socket states while initializing them in a far less-constrained manner. We use a powerful segmentation tool [37], textureless CAD models of our parts, and a state-of-the-art pose estimator [38] to estimate the 6-DOF poses of each part from RGB-D images. **Figure S11** shows our pipeline; for details, see **Appendix F**.

References

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

[2] K. Kimble, J. Albrecht, M. Zimmerman, and J. Falco. Performance measures to benchmark the grasping, manipulation, and assembly of deformable objects typical to manufacturing applications. *Frontiers in Robotics and AI*, 2022.

[3] B. Fu, S. K. Leong, X. Lian, and X. Ji. 6D robotic assembly based on RGB-only object pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

[4] B. Fu, S. K. Leong, Y. Di, J. Tang, and X. Ji. LanPose: Language-instructed 6D object pose estimation for robotic assembly. *arXiv preprint arXiv:2310.13819*, 2023.

[5] A. S. Morgan, B. Wen, J. Liang, A. Boularias, A. M. Dollar, and K. Bekris. Vision-driven compliant manipulation for reliable, high-precision assembly tasks. In *Robotics: Science and Systems*, 2021.

[6] B. Wen, W. Lian, K. Bekris, and S. Schaal. You Only Demonstrate Once: Category-level manipulation from single visual demonstration. In *Robotics: Science and Systems*, 2022.

[7] Y. Tian, J. Xu, Y. Li, J. Luo, S. Sueda, H. Li, K. D. Willis, and W. Matusik. Assemble Them All: Physics-based planning for generalizable assembly by disassembly. *ACM Transactions on Graphics*, 2022.

[8] Y. Tian, K. D. Willis, B. A. Omari, J. Luo, P. Ma, Y. Li, F. Javid, E. Gu, J. Jacob, S. Sueda, et al. ASAP: Automated sequence planning for complex robotic assembly with physical feasibility. *arXiv preprint arXiv:2309.16909*, 2023.

[9] O. Spector and D. Di Castro. InsertionNet: A scalable solution for insertion. *IEEE Robotics and Automation Letters*, 2021.

[10] O. Spector, V. Tchuiev, and D. Di Castro. InsertionNet 2.0: Minimal contact multi-step insertion using multimodal multiview sensory input. *arXiv preprint arXiv:2203.01153*, 2022.

[11] T. Z. Zhao, J. Luo, O. Sushkov, R. Pevceviciute, N. Heess, J. Scholz, S. Schaal, and S. Levine. Offline meta-reinforcement learning for industrial insertion. In *International Conference on Robotics and Automation (ICRA)*, 2022.

[12] L. H. De Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. In *IEEE International Conference on Robotics and Automation*, 1989.

[13] Wikipedia. T-symmetry, 2024.

[14] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2017.

[15] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[16] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

[17] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Neural Information Processing Systems*, 2017.

[18] X. B. Peng, P. Abbeel, S. Levine, and M. Van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics*, 2018.

[19] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. *Robotics: Science and Systems*, 2020.

[20] R. Bellman and R. Kalaba. On adaptive control processes. *IRE Transactions on Automatic Control*, 1959.

[21] P. Kidger, P. Bonnier, I. Perez Arribas, C. Salvi, and T. Lyons. Deep signature transforms. *Neural Information Processing Systems*, 2019.

[22] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

[23] L. G. Gyurkó, T. Lyons, M. Kontkowski, and J. Field. Extracting information from the signature of a financial data stream. *arXiv preprint arXiv:1307.7244*, 2013.

[24] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox. MimicGen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning*, 2023.

[25] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 2005.

[26] C. Eppner, A. Mousavian, and D. Fox. ACRONYM: A large-scale grasp dataset based on simulation. In *IEEE International Conference on Robotics and Automation*, 2021.

[27] B. Tang, M. A. Lin, I. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. Narang. IndustReal: Transferring contact-rich assembly tasks from simulation to reality. In *Robotics: Science and Systems*, 2023.

[28] D. Makoviichuk and V. Makoviychuk. rl-games: A high-performance framework for reinforcement learning, 2022.

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

[30] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[31] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac Gym: High performance GPU-based physics simulation for robot learning. In *Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

[32] M. Cuturi and M. Blondel. Soft-DTW: A differentiable loss function for time-series. In *International Conference on Machine Learning*, 2017.

[33] L. Barcelos, T. Lai, R. Oliveira, P. Borges, and F. Ramos. Path signatures for diversity in probabilistic trajectory optimisation. *arXiv preprint arXiv:2308.04071*, 2023.

[34] K.-T. Chen. Integration of paths–A faithful representation of paths by noncommutative formal power series. *Transactions of the American Mathematical Society*, 1958.

[35] T. J. Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 1998.

[36] P. Kidger and T. Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations*, 2021.

[37] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[38] B. Wen, W. Yang, J. Kautz, and S. Birchfield. FoundationPose: Unified 6D pose estimation and tracking of novel objects. *CVPR*, 2024.

[39] K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[40] Y. Narang, K. Storey, I. Akinola, M. Macklin, P. Reist, L. Wawrzyniak, Y. Guo, A. Moravanszky, G. State, M. Lu, et al. Factory: Fast contact for robotic assembly. In *Robotics: Science and Systems*, 2022.

[41] M. Bauza, A. Bronars, and A. Rodriguez. Tac2Pose: Tactile object pose estimation from the first touch. *The International Journal of Robotics Research*, 2023.

[42] B. Tang, M. A. Lin, I. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. Narang. IndustRealSim, 2024.

[43] B. Tang, M. A. Lin, I. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. Narang. IndustRealLib, 2024.

[44] R. Y. Tsai and R. K. Lenz. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 1989.

[45] A. Grunnet-Jepsen, J. Sweetser, T. Khuong, S. Dorodnicov, D. Tong, O. Mulla, H. Eliyahu, and E. Raikhel. Intel RealSense self-calibration for D400 series depth cameras, 2023.

[46] A. Grunnet-Jepsen, J. N. Sweetser, and J. Woodfill. Tuning depth cameras for best performance, 2023.

[47] A. Grunnet-Jepsen and D. Tong. Depth post-processing for Intel RealSense depth camera D400 series, 2023.

[48] Y. Labbé, J. Carpentier, M. Aubry, and J. Sivic. CosyPose: Consistent multi-view multi-object 6D pose estimation. In *European Conference on Computer Vision*, 2020.

[49] Y. Labbé, L. Manuelli, A. Mousavian, S. Tyree, S. Birchfield, J. Tremblay, J. Carpentier, M. Aubry, D. Fox, and J. Sivic. MegaPose: 6D pose estimation of novel objects via render & compare. In *Conference on Robot Learning*, 2022.

[50] X. Zhao, W. Ding, Y. An, Y. Du, T. Yu, M. Li, M. Tang, and J. Wang. Fast segment anything. *arXiv preprint arXiv:2306.12156*, 2023.

[51] C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee, and C. S. Hong. Faster Segment Anything: Towards lightweight SAM for mobile applications. *arXiv preprint arXiv:2306.14289*, 2023.