Meta-Reinforcement Learning for Robotic Industrial Insertion Tasks

Gerrit Schoettler[1], Ashvin Nair[2], Juan Aparicio Ojea[1],
Sergey Levine[2], Eugen Solowjow[1]

Abstract— Robotic insertion tasks are characterized by contact and friction mechanics, making them challenging for conventional feedback control methods due to unmodeled physical effects. Reinforcement learning (RL) is a promising approach for learning control policies in such settings. However, RL can be unsafe during exploration and might require a large amount of real-world training data, which is expensive to collect. In this paper, we study how to use meta-reinforcement learning to solve the bulk of the problem in simulation by solving a family of simulated industrial insertion tasks and then adapt policies quickly in the real world. We demonstrate our approach by training an agent to successfully perform challenging real-world insertion tasks using less than 20 trials of real-world experience.

I. Introduction

How can we embed prior knowledge into robot control systems? For simple tasks, an engineer can embed the entire solution into the system by instructing desired joint angle configurations for a robot to follow. Approaches for more complicated tasks might embed physical modelling into the control system, however this is often brittle because many real-world physics effects are difficult to capture accurately.

In this paper we consider the family of industrial insertion tasks where the robot inserts a grasped part into a tight-fitting socket. Today, the engineering time for fine-tuning state-of-the-art feedback controllers for such tasks can be similar in cost to the robot hardware itself. Flexible manufacturing with smaller lot-sizes and faster engineering cycles requires being able to quickly synthesize robust control policies, which can handle variability. This also broadens the space of manufacturing tasks accessible to robot automation. Notably, while the family of insertion tasks share significant structure, few existing methods have demonstrated the capability to take advantage of that similarity. Many of the most effective current methods for compliant robotic insertion [1], [2], [3], [4] require physical models, or else rely on manually-tuned feedback controllers to attain satisfactory performance.

Ideally, the task structure for an insertion task should be automatically inferred from the experience of having solved similar tasks. This insight leads us to meta-reinforcement learning methods, which given experience with a *family* of tasks, adapt to a new task from this family. However, while reinforcement learning (RL) methods can solve a task by learning from data, applying RL in the real world on many tasks is expensive. To circumvent this problem, we represent a task distribution entirely in simulation. Here,

we can control various facets of the environment, samples are cheap, and reward functions are easy to specify. In simulation, we learn the latent structure of the task using probabilistic embeddings for actor-critic RL (PEARL), an off-policy meta-RL algorithm, which embeds each task into a latent space [5]. The meta-learning algorithm first learns the task structure in simulation by training on a wide variety of generated insertion tasks. For our family of insertion tasks, the size and placement of the components, parameters of the controller, and magnitude of sensor errors are all randomized, resulting in the policy learning robust and adaptive search strategies based on the latent task code. After training in simulation, the algorithm is then capable of quickly adapting to a real-world task.

In this work, we solve industrial robotic insertion problems by learning the latent structure of the tasks with meta-RL. Concretely, we look at the task of grasping and inserting two parts: a Misumi E-model electrical connector (one of the most challenging tasks from the IROS 2017 Robotic Grasping and Manipulation Competition [6]) and a gear on a shaft. We adapt the same policy, which was learned in simulation, to each of the two tasks, despite their distinct physical properties. Moreover, in each task, our method adapts with just 20 trials, significantly fewer than in previous work. We present the robotic system, including a system to account for grasp errors from camera images. Finally, we cover the comprehensive evaluation of our method against both conventional methods and learning-based methods for different degrees of environment variability.

II. Related Work

Studies on peg-in-hole insertion have been ubiquitous in industrial automation, as it is representative of many common assembly problems. The key challenges involved in insertion tasks are modeling of physical contacts, and handling errors in perception and control. Since physical modeling of contacts and friction is often difficult, deployed controllers for insertions are based on heuristic search patterns that handle the issues implicitly. These methods include random search, spiral search or raster search [3]. The search patterns are combined with compliant control methods, which have been amongst the first model-based strategies for solving insertion tasks [1], [2], [4]. The parameters of these controllers are manually tuned in order to overcome perception and control errors for a specific system. The search patterns are often embedded in control state-machines, which guide the system. Other approaches focus on high-precision assembly by taking advantage of high-dimensional geometry or contact

[1]Siemens {gerrit.schoettler, juan.aparicio, eugen.solowjow}@siemens.com
[2]UC Berkeley {anair17, levine}@berkeley.edu

1. Design family of simulated tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$

2. Learn latent embedding of task context $q_\phi(\boldsymbol{z}|\boldsymbol{c})$ and policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ on family of simulated tasks using PEARL [5]
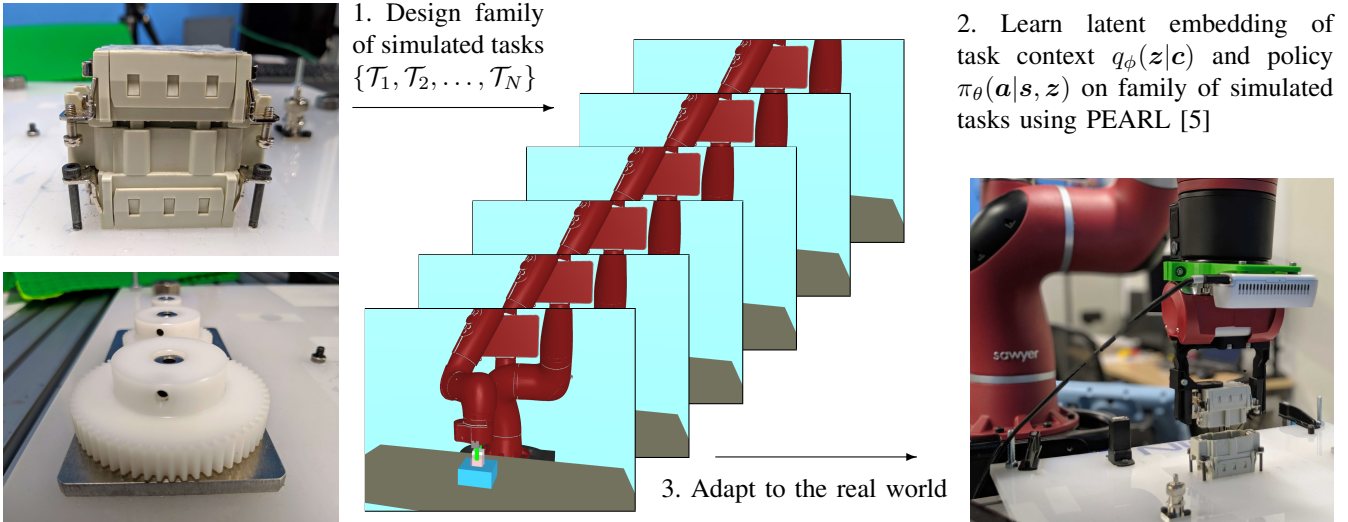
3. Adapt to the real world

Fig. 1: We present results on solving two real-world use cases of robotic industrial insertion tasks: plugging in an electrical connector and a gear assembly task, both shown on the left. We model a family of simulated insertion tasks by randomizing simulator parameters. Next, we use meta-reinforcement learning in simulation to learn a latent embedding of tasks and a policy that can rapidly adapt to a new task in that family. Finally, we show that the policy can indeed be quickly adapted to real-world tasks with only 20 trials on the physical robot. Videos are available at `http://pearl-insertion.github.io`.

information [7], [8], [9], [10]. These approaches require a significant amount of engineering, modeling and tuning.

Instead of relying on human ingenuity to solve robotic control tasks, the paradigm of RL has promise to autonomously learn the control policy from data. RL has thus far been used in a variety of settings, such as playing ping-pong [11]. RL with expressive function approximators, or deep RL, further allows tasks to be learned from raw sensor inputs such as images. Deep RL has shown success in games [12], [13], in learning fine robotic manipulation skills [14] and navigation [15]. Specifically, peg insertion tasks have commonly been considered in deep RL settings. Florensa et al. investigate difficult insertion tasks with sparse rewards in simulation using a reverse curriculum [16]. Another approach to solving these tasks is to use prior data such as demonstrations [17], [18], [19], [20]. Vecerik et al. [21] perform a real-world insertion task utilizing demonstrations. Alternatively, one can learn a residual policy for contacts that is superposed with conventional controls [22], [23].

Another line of work considers first learning on simulation models of a task and then transferring the policy to the real world. One approach is domain randomization, which trains on a wide distribution of tasks in simulation assuming that the real world task is captured in that distribution [24], [25], [26], [27]. Further work adaptively randomizes the distribution of the tasks [28], [29], [30]. One can also actively adapt the simulator by switching between simulation and real-world interaction to guide the simulator [31], [32]. In this work, we take a different approach by using meta-learning to learn a distribution of skills in simulation, followed by an adaptation in the real world.

III. BACKGROUND

In this section, we define basic notation and describe reinforcement learning and meta-learning.

A. Reinforcement Learning

We model our problem as a Markov decision process, where an agent at every discrete timestep t is in state $\mathbf{s}_t \in \mathcal{S}$, executes an action $\mathbf{a}_t \in \mathcal{A}$, receives a scalar reward $r_t(\mathbf{s}_t, \mathbf{a}_t)$, and the environment evolves to the next state according to the transition probability $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. The agent attempts to maximize the expected return $R = \sum_{t=0}^{H} \gamma^t r_t$ where H is the planning horizon and $\gamma \in (0, 1)$ is a discount factor. In reinforcement learning, the agent learns a policy $\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$ that is optimized from data.

B. Meta-Learning

Meta-learning is the problem of rapid adaptation: given experience in some family of tasks, how can use that experience to quickly adapt to a new task at test time? Formally, consider a task $\mathcal{T} = \{r(\mathbf{s}_t, \mathbf{a}_t), p(\mathbf{s}_1), p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\}$ to be defined by the reward function $r_t(\mathbf{s}_t, \mathbf{s}_t)$, initial state distribution $p(\mathbf{s}_1)$, and transition distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. We consider some distribution over tasks $p(\mathcal{T})$, which we want to perform well on at test time by collecting limited experience during training time.

Several methods have explored this setting. One class of methods separates the training time into meta-training and meta-testing, and attempts to learn a model (a policy, forward model, or loss function) during meta-training that improves meta-test performance [33], [34], [35], [36], [37]. In the meta-RL setting, these methods perform back-propagation

with on-policy gradient updates, which limits their sample efficiency.

The other class of methods effectively learn a latent representation of the task [38], [39], [5]. The last one of these, PEARL by Rakelly et al., can take advantage of off-policy data, allowing sample-efficient learning on real robots, and we describe the algorithm further in the following section.

C. Probabilistic Embeddings for Actor-Critic RL

Probabilistic embeddings for actor-critic RL (PEARL) is a meta-learning algorithm that enables sample efficient meta-learning by reusing past data with off-policy RL algorithms [5]. The key idea is to condition the policy on the past transitions of the current task, which is termed the context $c_{1:n}^\tau$. The context is encoded into a latent variable z, and we train the policy $\pi_\theta(a|s, z)$. During meta-training PEARL, learns the policy parameters and the inference network $q_\phi(z|c)$ which is factorized as $q_\phi(z|c_{1:N}) = \prod_{n=1}^N \Psi(z|c_n)$ and Ψ are Gaussian factors, resulting in a Gaussian posterior. The parameters θ and ϕ are learned with an off-policy algorithm that additionally learns a critic. At meta-test time, $z \sim q(z|c)$ is sampled before every rollout, and the new data is used to update the posterior. The training of PEARL is outlined in Alg. 1.

IV. INDUSTRIAL INSERTION TASKS

We apply meta-learning to two real-world industrial insertion tasks, a waterproof electrical connector plug and a 3D-printed gear, pictured in Fig. 1. For our experiments, we use the Rethink Robotics Sawyer robot running a Cartesian-space end effector position controller, further detailed in V-C.1. Thus the action-space is 3-dimensional. As observations, the current end effector positions relative to the the assumed goal location are used, resulting in a 3-dimensional observation space. Each real-world trial consists of 50 steps with a maximum step size of $2\,\mathrm{mm}$. The duration ΔT of each step is calculated by multiplying the length of the step with a desired average velocity of $0.01\,\mathrm{m/s}$. After each trial, a reset is performed, the reset position is located $5\,\mathrm{mm}$ above the insertion socket. The workspace of the robot is defined as a cylinder with a radius of $3\,\mathrm{cm}$ and a height of $4\,\mathrm{cm}$, centered at the goal location. If it happens that the robot leaves the workspace, it gets pushed back inside, perpendicular to the nearest surface of the cylinder. If an insertion is completed before the end of a trial, the end effector is kept still but rewards are collected until the end of the trial. We use the following sparse reward function during the real-world adaptation phase:

$$r_t = \begin{cases} 1 & \text{if height} \le \text{threshold} \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where the threshold to detect a successful insertion is $5\,\mathrm{mm}$ below the tip of the insertion.

V. METHOD

The key insight of this work is that industrial insertion tasks have shared structure that can be exploited by learning from data on a family of tasks. Thus, in order to obtain a general meta-RL policy for the real-world, we first design a family of tasks in simulation to reflect the real world tasks. Then, we use meta-learning in simulation to learn a policy and task embedding that allows fast adaptation to new tasks in that family. Finally, we apply the learned policy in the real world, where the complete task is to first grasp a part and then insert it. Below, we describe each of these steps in detail.

A. Simulated Environment Design

To simulate the family of industrial insertion tasks, we use the physics engine MuJoCo [40]. The simulated environment, shown in Fig. 2, contains the Sawyer robot, a table, immovable blocks on the table that form a hole, and a block, that fits into this hole, attached to the robot's gripper. Cartesian-space end effector position control is used, with the maximum step size set to $2\,\mathrm{mm}$. The family of tasks is generated by randomizing the following simulation parameters:

- O, the horizontal offset of the goal, within $\pm 5\,\mathrm{mm}$.
- C, the clearance of the insertion task, modified by changing the size of the block between $13\,\mathrm{mm}$ and $14\,\mathrm{mm}$, while the size of the square hole remains fixed at $15\,\mathrm{mm}$.
- S, the scaling of the position controller's step size, in the range $\pm 10\%$.

Additionally, the reset position of the end-effector is uniformly sampled inside a cube with side length $5\,\mathrm{mm}$, located $5\,\mathrm{mm}$ above the ideal goal, before each reset.
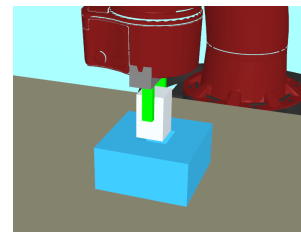

Fig. 2: In the simulated environment, we model the connector insertion with a square block that is inserted into a slightly larger square hole. The parameters of the simulator are randomized to generate a family of insertion tasks of different difficulties.

The observation space of the simulated environment consists of the 3-dimensional end effector location, measured relative to the ideal goal to which the random perturbations are added. The reward function during the simulated meta-training is the ℓ_2-distance to the current goal. During the meta-adaptation phase, the sparse reward function of Eq. (1) is used. This is done because the exact goal location is not known during test-time, but an insertion is detected via a height measurement. The choice of rewards during training and test time are comparable to prior work [5].

B. Sim-to-Real Transfer via Meta Reinforcement Learning

Using the simulator, we train a policy with the meta-RL algorithm on the family of tasks. Although any meta-RL algorithm could potentially be used, in this work we use PEARL for a number of reasons. First, due to its capability for off-policy training, it is highly sample efficient. Second, PEARL learns a task embedding, which allows it to explicitly learn a latent structure over the family of tasks. This property of the algorithm also allows for very fast adaptation, which is vital in the real-world as collecting real-world samples can be expensive. The meta-RL policy trained in simulation is then able to adapt to tasks sampled from the training distribution within a small number of trials.

We then perform policy adaptation on the real system until consistent performance is reached, as detailed in Alg. 2. From the perspective of the algorithm, the real system is just another task to be adapted to. This simple procedure is surprisingly effective at learning robust, adaptive controllers.

C. Real-World Execution

While we only train the insertion skill in simulation, in the real world the task is to first grasp the part and then insert it. In this section, we cover the real-world implementation details including a more accurate controller for the Sawyer, and an algorithm for grasp detection and correction.

1) Robot Impedance Controller: The control scheme we developed for precise end effector position control of the Sawyer robot is presented in Alg. 3. With this controller, the robot consistently reaches a target with a precision of $0.1\,\text{mm}$. In addition to the low-level control, we added a non-interfering high-level impedance controller, that does not decrease precision. We send velocity commands to the robot, using position commands instead resulted in an average position error of $0.4\,\text{mm}$.

To safely perform insertion tasks, we developed an impedance controller that operates in end effector position space. After each execution of Alg. 3, the vertical force at the end effector is measured. If it exceeds a threshold of $6\,\text{N}$, a small upwards move is initiated. If the force still exceeds the threshold, a $0.1\,\text{mm}$ larger upwards move follows. An additional safety feature, included in Alg. 3, is a low frequency measurement of the end effector forces in-between the high frequency commands that are sent to the robot. When a threshold of $\mathbf{f}_{\max} = 10\,\text{N}$ is exceeded, the robot stops the current motion and waits for the next commanded action. During upwards movements, this safety feature is disabled to prevent the robot from getting stuck while pressing down.

2) Grasp Algorithm: A RealSense D435 depth camera is mounted to the robot arm and used to scan the workspace to calculate a grasp based on a depth image, as shown in Fig. 3. We clean the depth image from artifacts and use a hand-tuned distance threshold to binarize the image. We then apply a contour finding algorithm to extract rectangular contours, check if the size of a found contour matches the assumed object size, and use temporal filtering to average the object

---

Algorithm 1 PEARL Training in Simulation

Require: Batch of simulated training tasks $\{\mathcal{T}_i\}_{i=1\dots T}$ from $p(\mathcal{T})$, learning rates $\alpha_1, \alpha_2, \alpha_3$
1: Initialize replay buffers $\mathcal{B}^i$ for each training task
2: **while** not done **do**
3:      **for** each $\mathcal{T}_i$ **do**
4:          Initialize context $\mathbf{c}^i = \{\}$
5:          **for** $k = 1, \dots, K$ **do**
6:             Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
7:             Gather data from $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$ and add to $\mathcal{B}^i$
8:             Update $\mathbf{c}^i = \{(\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j)\}_{j:1\dots N} \sim \mathcal{B}^i$
9:             Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
10:          **end for**
11:      **end for**
12:      **for** step in training steps **do**
13:          **for** each $\mathcal{T}_i$ **do**
14:          Sample context $\mathbf{c}^i \sim \mathcal{S}_c(\mathcal{B}^i)$ and RL batch $b^i \sim \mathcal{B}^i$
15:          Sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{c}^i)$
16:          $\mathcal{L}^i_{actor} = \mathcal{L}_{actor}(b^i, \mathbf{z})$
17:          $\mathcal{L}^i_{critic} = \mathcal{L}_{critic}(b^i, \mathbf{z})$
18:          $\mathcal{L}^i_{KL} = \beta D_{\text{KL}}(q(\mathbf{z}|\mathbf{c}^i)\|r(\mathbf{z}))$
19:          **end for**
20:      $\phi \leftarrow \phi - \alpha_1 \nabla_\phi \sum_i (\mathcal{L}^i_{critic} + \mathcal{L}^i_{KL})$
21:      $\theta_\pi \leftarrow \theta_\pi - \alpha_2 \nabla_\theta \sum_i \mathcal{L}^i_{actor}$
22:      $\theta_Q \leftarrow \theta_Q - \alpha_3 \nabla_\theta \sum_i \mathcal{L}^i_{critic}$
23:      **end for**
24: **end while**

Algorithm 2 PEARL Sim-to-Real Adaptation

Require: Trained Meta-RL poliy π_θ, trained context encoder q_ϕ, real test task $\mathcal{T}$
1: **for** $k = 1, \dots, K$ **do**
2:      Sample $z \sim q_\phi(\mathbf{z}|c^{\mathcal{T}})$
3:      Roll out policy $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{z})$ to collect data $D^{\mathcal{T}}_k = \{(\mathbf{s}_j, \mathbf{a}_j), \mathbf{s}'_j, r_j\}_{j:1,\dots,N}$
4:      Accumulate context $\mathbf{c}^{\mathcal{T}} = \mathbf{c}^{\mathcal{T}} \cup \mathbf{D}^{\mathcal{T}}_k$
5: **end for**

Algorithm 3 Robot Control Scheme

Require: desired end effector location, orientation, and duration for action ΔT
1: Calculate desired joint angles γ_{des} via inverse kinematics
2: Form smooth spline $\mathbf{S}$ between γ_0 and γ_{des}
3: **while** $t < t_0 + \Delta T$ **do**
4:      Evaluate $\nabla\gamma_t$ as slope of $\mathbf{S}$ at time t
5:      Send $\nabla\gamma_t$ as joint velocity command to actuators
6:      Measure end effector forces $\mathbf{f}_t$ at $10\,\text{Hz}$
7:      **if** any $\mathbf{f}_t > \mathbf{f}_{\max}$ **then**
8:          break
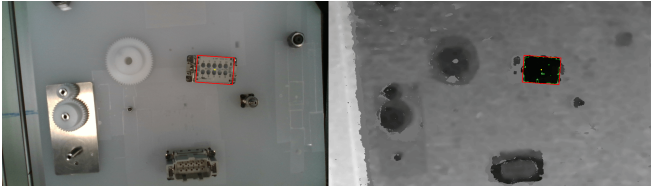9:   **end if**
10: **end while**

Fig. 3: Robot view through a RealSense D435 camera, mounted to the robot arm. The depth image is binarized with a tuned threshold and contours are extracted. A rectangular bounding box (red) is drawn for contours of the expected size and used to calculate an aligned and centered grasp. Averaging the calculated grasps over over multiple frames improved the stability, allowing 100 consecutive successful grasps.

location over multiple frames. The grasp will be planned along one of the principal axes of the rectangular bounding box. Hand-eye calibration is used to find the corresponding real-world coordinates in the robot frame. The requirements for this grasp approach are that the graspable object is clearly detectable in the depth image and that the distance threshold and the assumed object size are set appropriately.

3) Grasp Error Correction: In a real factory setting, each object that is about to be inserted by a robot needs to be grasped first. This increases the time per insertion attempt and induces unavoidable grasp errors when using a non-self-centering parallel gripper. In order to resemble the real setting as precisely, as possible, we include the grasping in our experimental setup. To mitigate grasp errors, we propose a grasp-correction algorithm that only requires a single image to calculate the object's displacement with respect to a reference grasp.

Our grasp correction algorithm uses an image of the bottom of the grasped object, as shown in Fig. 4, and compares it with a reference image using cross-correlation. From the cross-correlation of the new image with the reference image, the translation with respect to the reference grasp pose can be inferred reliably. The goal location is then adjusted based on the computer grasp error.

Rotational grasp errors are not considered because a the objects were not seen to rotate inside the parallel gripper and the rotation of the fixed goal location was calibrated. In different setups however, rotations may be a major source of error and should be investigated.

VI. EXPERIMENTAL RESULTS

We conduct a series of experiments to answer the following questions:

A. Can PEARL learn to robustly adapt to novel insertion tasks in simulation?

B. Is it possible to adapt insertion policies learned using meta-RL in simulation to the real reward?

C. How does sim-to-real meta-RL compare to existing solutions to robotic insertion problems, in terms of robustness and efficiency?
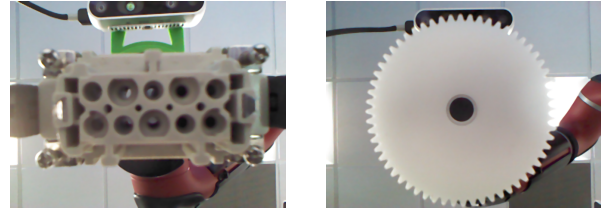

Fig. 4: Camera view when scanning the bottom of the part to calculate the pixel offset with respect to a reference grasp. A Kinect v1 camera is mounted upside down on the table to take an image after each grasp.

D. What patterns and behavior does the algorithm learn in simulation that allow it to transfer to the real world?

We address each of these questions in our experimental evaluation, presented below.

A. Adaptation in Simulation

First, we examine the performance of PEARL on our family of simulated insertion tasks. The adaptation performance on test tasks after training is shown in Fig 5.
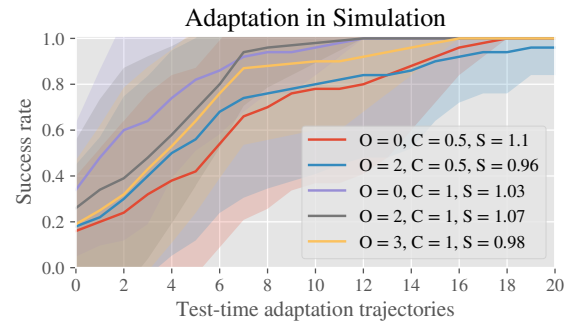

Fig. 5: Success rate on test tasks in the adaptation phase in simulation after training. Per experiment, 20 random seeds are evaluated. We see that the PEARL policy successfully learned to adapt to unseen task in simulation. Descriptions of the randomized parameters are given in Section V-A.

In our simulation, we see that the zero-shot performance of the trained policies is about 20%. But given 20 trials in the new environment, the algorithm can successfully adapt to solve each of the new tasks.

B. Real-World Adaptation

After training in simulation, we adapt the meta-RL policy to tasks in the real world. As discussed in Sec. V-C.3, in the real-world tasks the object (either the connector or the gear) is picked up using our grasp system and grasp errors are compensated according to our grasp correction algorithm.

In addition to the grasping, we consider robustness to poorly calibrated setups by perturbing the goal location. Thus, we evaluate the method on five different tasks between the two use cases: the plug insertion task with no noise, ± 2mm noise, and ± 3mm noise, and the the gear task with no noise, and ± 2mm noise.

The real-world adaptation results are presented in Fig. 6. The results show that, in each case, we can adapt to all the tasks in less than 20 trials of real-world interaction. The goal perturbation presents a challenge to all methods.
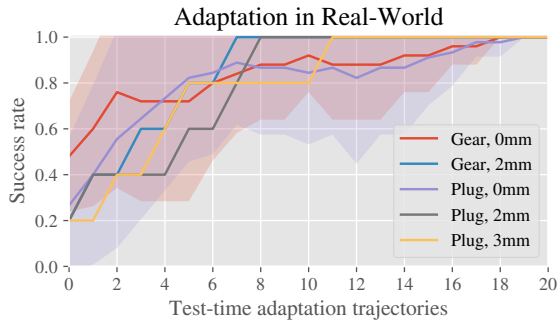

Fig. 6: Comparison of the success rates after each trial of the adaptation phase in the real world. Note that in each of the five tasks, our method is able to adapt to the task with less than 20 real-world trials.

We see that our methods is able to still solve the insertion task. In contrast to the heuristic methods, our method has the capacity to learn very complex search strategies and to continuously adapt. Since the simulated meta-training phase included randomization of the goal location, the meta-trained policy explores quite broadly when adapting to the real world.

C. Comparison of Robustness and Sample Efficiency

In these experiments, we evaluate whether meta-RL is a viable solution for industrial insertion tasks, comparing the method to existing solutions that are used today. We compare to four baselines, which are either covered in past research or used widely in industry. In total, we compare the following methods:

1) **Straight downwards.** Move straight downwards from the reset position.
2) **Random search.** In this stochastic search policy, described in [41], the robot moves horizontally between search points that are sampled uniformly inside a square with side length $6\,\mathrm{mm}$, centered above the assumed goal location. The robot moves downwards to the first sampled point until a vertical contact force of $3\,\mathrm{N}$ is sensed. If no successful insertion is detected, the end effector moves back upwards until the measured force decreases, then moves to the next sampled point where it attempts another insertion in the same way. At most 50 random insertion attempts are executed in each trial.
3) **Spiral search.** The robot generates a spiral above the assumed goal location and iteratively attempts to insert at points along the spiral path [41]. During the downwards movement, a force threshold of $3\,\mathrm{N}$ is used to indicate contact in the same way as described above in random search. The radius of the spiral increases linearly by $0.5\,\mathrm{mm}$ per rotation and insertions are attempted at

TABLE I: Comparison of real-world insertion performance out of 25 trials. Only the policy trained with PEARL can achieve a very high success rate on all tasks. In the bottom table, we show the average time for successful insertions.

Success Rate

Policy	Plug			Gear	
	0	±2mm	±3mm	0	±2mm
Straight Down	0.88	0.0	0.0	0.32	0.0
Random Search	**1.0**	**1.0**	**1.0**	0.48	0.32
Spiral-Search	**1.0**	**1.0**	**1.0**	0.84	0.8
RL from Scratch	**1.0**	0.32	0.0	**1.0**	0.92
PEARL Sim2Real	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**

Insertion Time [s]

Policy	Plug			Gear	
	0	±2mm	±3mm	0	±2mm
Straight Down	**3.3** ±0.6	–	–	**5.3** ±1.4	–
Random Search	5.6 ±2.0	**7.0** ±2.3	**9.7** ±4.6	20.3 ±7.3	23.3 ±9.3
Spiral-Search	6.0 ±4.7	13.6 ±6.6	26.6 ±4.8	8.0 ±2.6	17.3 ±5.2
RL from Scratch	11.7 ±4.2	–	–	**5.8** ±0.7	–
PEARL Sim2Real	5.3 ±2.7	**6.8** ±4.7	**8.2** ±4.5	**5.7** ±2.6	**8.0** ±5.5

points $45\,\mathrm{deg}$ apart. At most, the robot moves through 50 points.

4) **RL from scratch.** We train SAC [42] in the real world from scratch, using the same action space, state space and sparse reward function as in the real-world adaptation phase with PEARL, described in Sec. IV. The training with SAC requires substantially more environment steps than adapting PEARL, which is why we choose to rigidly mount the adapter to the robot's gripper and leave out the grasping during the training. At test-time, the grasping is performed. The SAC policies were trained for 2 hours and 20 minutes; repeating success was already visible after 1 hour and 20 minutes of training.
5) **PEARL Sim2Real.** Our method using meta RL, as described in Section V.

We evaluate these methods along two dimensions. Most importantly, we measure the success rate of the method on a task. We also measure the time needed for each insertion, to compare the efficiency of the different methods. The measurement of efficiency is important for practical applications, since throughput is a major consideration in industrial settings. Results of the experiments performed on all five tasks are reported in Tab. I. We see that our method is the only one that consistently solves every task, and is almost always the fastest, except when moving straight down works. The gear use-case is visibly more difficult and not

TABLE II: Deterministic and stochastic policy evaluation, 25 trials each. It was observed that a deterministic policy is equally able to adapt to the real-world setting, but shows slightly less consistency at test-time. When getting stuck, the deterministic policy predominantly failed to recover, whereas the stochastic policy managed to still solve the task.

PEARL Success Rate	Deterministic	Stochastic
Connector Plug +3mm	0.44	**1.0**
Gear +2mm	0.84	**1.0**

solvable with naïve downwards movement. The two heuristic search methods: random search and spiral search, are not always able to succeed at the more difficult settings in the given 50 steps. Meta-RL sim-to-real transfer shows the best performance among the most difficult tasks. Videos of our results can be viewed at `http://pearl-insertion.github.io`.

D. What behavior does the policy transfer from simulation?

We believe the main knowledge transferred from sim-to-real is *structured exploration noise*. We investigate this by comparing the learned stochastic policy of PEARL to its deterministic version, done by always choosing the most likely action. The stochastic latent variable z remains stochastic though. Prior work has consistently found that, although stochasticity helps at training time, the deterministic policy gives better final returns [42]. In Fig. 7 and Tab. II, we compare the stochastic and deterministic policy when learning in simulation and performing sim-to-real transfer with PEARL. In our simulation, the stochastic policy consistently achieves a higher success rate. During the real-world adaptation, we observed better exploration with the stochastic policy, as well as a better final performance, reported in Tab. II. The failed insertion attempts of the deterministic policy happened because the gear got stuck at the first stage of the insertion. The stochastic policy was still able to recover in all cases because it produced oscillating movements in that case. In Fig. 8, we visualize the computed actions of a PEARL policy that was trained on a 2D sparse point robot environment with uniformly distributed goals around the origin. It is visible that the deterministic policy does not perform any movement inside of the goal region, whereas the stochastic policy learned to fully explore the goal region. We observed this movement inside of the goal region to be beneficial when performing insertions in the real world, as a slight misjudgement of the shape, size and location of the goal region can be compensated with these stochastic actions.

Finally, we can infer what behavior is learned by analyzing the situations in which the sim-to-real transfer with PEARL did not work well. For instance, the real-world adaptation failed when the randomization of the reset position was left out during the training in simulation. Then, the trained meta-policy did not learn a stable behavior outside of the direct paths to the training goals. In the real world adaptation phase, inaccuracies of the real robot's movement caused the end effector to enter unstable regions, in which a continuous
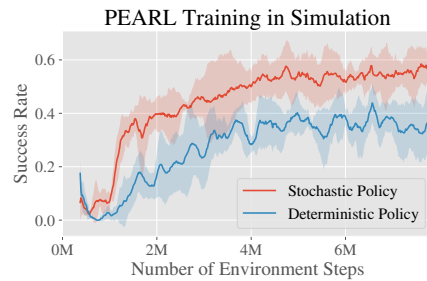

Fig. 7: Comparison of the PEARL training in simulation with either a deterministic or stochastic policy. Although the deterministic policy is formed by computing the maximum likely action, our model of the insertion task benefits from a stochastic policy evaluation. We think this is due to the more extensive exploration during the adaptation step of PEARL.
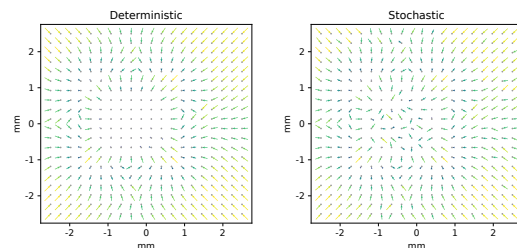

Fig. 8: Visualization of the policy outputs on a grid around a goal. Here, a policy was trained on a 2D sparse point robot environment. In contrast to a deterministic policy evaluation, a stochastic evaluation also shows movement inside of the goal region, which we observed as beneficial when performing insertion tasks with a real robot.

movement away from the goal occurred. The real-world adaptation also failed when the amount of randomization was too high, as sometimes none of the insertion attempts during the real-world adaptation phase succeeded. Due to the use of sparse rewards, PEARL does not obtain any explicit information about the goal location in this case. When we observed this failure case, we reduced the amount of randomization in simulation.

VII. CONCLUSION

In this paper, we studied meta-reinforcement learning for industrial insertion tasks. Our method first performs meta-training in a low-fidelity simulation, and then actively adapts to a variety of real-world insertion and assembly tasks. This approach can solve complex real-world tasks in under 20 trials, performing connector assembly and a 3D-printed gear insertion task. We also demonstrated the feasibility of our method under challenging conditions, such as noisy goal specification and complex connector geometries.

Our method shifts the burden of engineering robotics solutions from designing accurate analytic physical models to designing a family of representative simulated tasks. Furthermore, as our method requires experience in the real world only for the final adaptation step, the work of designing the

simulation may be amortized across many tasks. Thus, we believe that our work illustrates the potential of meta-RL to provide a scalable and general method for rapid adaptation in manufacturing and industrial robotics.

VIII. Acknowledgements

References

[1] D. E. Whitney, "Force feedback control of manipulator fine motions," " 1977.

[2] ——, "Quasi-static assembly of compliantly supported rigid parts," *ASME J. Dynamic Systems Measurement, and Control*, vol. 104, pp. 65–77, 1982.

[3] S. R. Chhatpar and M.S. Branicky, "Search strategies for peg-in-hole assemblies with position uncertainty," in *IROS*, 2001.

[4] H. Park, J.-H. Bae, J.-H. Park, M.-H. Baeg, and J. Park, "Intuitive peg-in-hole assembly strategy with a compliant manipulator," in *IEEE ISR 2013*. IEEE, 2013, pp. 1–5.

[5] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *ICML*, 2019, pp. 5331–5340.

[6] J. Falco, Y. Sun, and M. Roa, "Robotic grasping and manipulation competition: Competitor feedback and lessons learned," in *Robotic Grasping and Manipulation*, Y. Sun and J. Falco, Eds. Cham: Springer International Publishing, 2018, pp. 180–189.

[7] R. Li, R. Platt, W. Yuan, A. Ten Pas, N. Roscup, M. A. Srinivasan, and E. Adelson, "Localization and Manipulation of Small Parts Using GelSight Tactile Sensing," in *IROS*, 2014.

[8] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the hindsight plan episodic mpc improvement," in *ICRA*, 2017, pp. 336–343.

[9] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *IROS*, 2017, pp. 819–825.

[10] J. Luo, E. Solowjow, C. Wen, J. Aparicio Ojea, A. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *ICRA*, 2019.

[11] J. Peters, K. Mülling, and Y. Altün, "Relative Entropy Policy Search," in *AAAI Conference on Artificial Intelligence*, 2010, pp. 1607–1612.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *NIPS Workshop on Deep Learning*, 2013, pp. 1–9.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, jan 2016.

[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research (JMLR)*, vol. 17, no. 1, pp. 1334–1373, 2016.

[15] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation," in *ICRA*, 2018.

[16] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, "Reverse Curriculum Generation for Reinforcement Learning," in *ICLR*, 2018.

[17] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Learning from Demonstrations for Real World Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2018.

[18] A. Nair, B. Mcgrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming Exploration in Reinforcement Learning with Demonstrations," in *ICRA*, 2018.

[19] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," in *Robotics: Science and Systems*, 2018.

[20] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. Aparicio Ojea, E. Solowjow, and S. Levine, "Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards," *arXiv preprint arXiv: 1906.05841*, 2019.

[21] M. Večerík, 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," *CoRR*, vol. abs/1707.0, 2017.

[22] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. Aparicio Ojea, E. Solowjow, and S. Levine, "Residual Reinforcement Learning for Robot Control," in *ICRA*, 2019.

[23] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual Policy Learning," *arXiv preprint arXiv:1812.06298*, 2018.

[24] F. Sadeghi and S. Levine, "CAD 2 RL: Real Single-Image Flight Without a Single Real Image," in *RSS*, 2017.

[25] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," *IROS*, 2017.

[26] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *ICRA*, 2018.

[27] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric Actor Critic for Image-Based Robot Learning," *RSS*, 2018.

[28] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's Cube with a Robot Hand," " oct 2019.

[29] F. Ramos, R. Carvalhaes Possas, and D. Fox, "BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators," in *Robotics: Science and Systems (RSS)*, 2019.

[30] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active Domain Randomization," in *Conference on Robot Learning (CoRL)*, apr 2019.

[31] W. Zhou, L. Pinto, and A. Gupta, "Environment Probing Interaction Policies," in *International Conference on Learning Representations*, 2019.

[32] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience," in *ICRA*, 2019.

[33] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *ICML*, 2017.

[34] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths, "Recasting Gradient-Based Meta-Learning As Hierarchical Bayes," in *ICLR*, 2018.

[35] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal Planning Networks," in *ICML*, 2018.

[36] H. Bharadhwaj, Z. Wang, Y. Bengio, and L. Paull, "A data-efficient framework for training and sim-to-real transfer of navigation policies," in *International Conference on Robotics and Automation (ICRA)*, vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., may 2019, pp. 782–788.

[37] M. Wortsman, K. Ehsani, M. Rastegari, A. Farhadi, and R. Mottaghi, "Learning to Learn How to Learn: Self-Adaptive Visual Navigation Using Meta-Learning," in *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[38] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL\$^2\$: Fast Reinforcement Learning via Slow Reinforcement Learning," " nov 2016.

[39] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *NeurIPS*, 2016, pp. 3988–3996.

[40] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IROS*, 2012, pp. 5026–5033.

[41] J. A. Marvel, R. Bostelman, and J. Falco, "Multi-Robot Assembly Strategies and Metrics," *ACM computing surveys*, vol. 51, no. 1, p. 14, 2018.

[42] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *ICML*, 2018.