SCALE: Causal Learning of Robot Manipulation Skills from Simulation

Author names withheld

Editor: Under Review for CLeaR 2022

Abstract

We present an approach for discovering and learning distinct types of robot skills for a given task based on the task parameters that are relevant to each skill. Our approach frames simulation as a causal inference engine, where interventions elicit the dependence between solving a manipulation task and context variables. Specifically, we learn Regional Causal Skills (RCS) which extends the robot skill framework by learning Data Generation Regions (DGR). A DGR is a region in the context space such that skills learnt in this region have the same set of relevant context variables. This set of relevant context variables parameterize the RCS and result from interventional queries of the simulator. Each skill can be viewed as a distinct behavior that can be executed by a given low-level robot controller. We demonstrate our approach for two representative manipulation tasks: block stacking and peg-in-hole insertion. Our experiments show that our approach yields diverse skills that are compact and suitable for sim-to-real transfer learning.

Keywords: robot skill learning, causal inference, structure learning

1. Introduction

In order for robots to perform useful work in unstructured environments such as homes, kitchens, and restaurants, it is critical for robots to learn a diverse set of skills that adapt to the current situation (Kroemer et al. (2021)). Different skills can be used to model distinct strategies for performing a given task. For example, to open a door, one skill could place a hand on the door and push it open while another could grasp the handle and pull it open. Another skill could also twist the handle before pulling the door open. In all cases, the door will be opened, but the approach to achieving this goal is different. Different sets of these strategies will work in different situations. Importantly, the set of relevant object parameters also vary between these strategies. For example, the pushing strategy does not need to consider the location of the handle while the pulling strategy does, and only for rotating handles does the robot need to consider the unblocking angle. Identifying the relevant variables for each strategy is important for achieving robust and effective learning.

We propose an approach for discovering distinct manipulation skills by first identifying task instances in which different sets of variables are considered to be relevant for performing the task. We use a vector of context variables to represent different task instances, e.g., different door geometries, and each skill represents a mapping from the context space to a set of low-level controller parameters for executing the task. Using a simulator as a causal inference engine, the robot begins by determining suitable controller parameters for individual task instances in simulation. It then applies interventions and causal reasoning to determine the sets of relevant context variables for each task instance. By combining together task instances with the same sets of relevant context variables, we create sets of distinct data generation regions. The samples from these regions are then used to learn a general parameterized skill that adapts to only the corresponding set of relevant context variables. Once a skill’s policy has been learned, we also learn its preconditions, which determine the
Figure 1: The figure shows an overview of the proposed framework. The robot is given a context space, control policy, task simulator, and task reward. The robot then samples a set of contexts to create task instances, which it subsequently solves for that instance. The robot then applies interventions on the context to identify skill-relevant parameters. Contexts with the same set of policy-relevant parameters are combined to form data generation regions. Each region is then used to learn a skill policy with the corresponding set of policy-relevant parameters. For each skill we subsequently learn a set of preconditions within the space of the precondition-relevant context parameters to determine where the skill can ultimately be applied. The pairs of policies and preconditions are then combined to create a skill library for completing the given task.

contexts in which the skill can actually be executed. The robot can then select the specific skill from its library with the highest predicted probability of success given the current context.

Prior work by Lee et al. on CREST (Lee et al. (2021)) has shown that identifying the set of relevant parameters for each skill allows the robot to learn the skill policies more efficiently, improve generalization along irrelevant context variations, and increase transfer performance when mapping from simulation to real robot execution. We extend upon this prior work by using CREST to discover a library of multiple distinct skills, each with their own parameterization of relevant context variables, for performing the task. In this manner the robot can learn different strategies for performing tasks, using simpler skills when possible and only incorporating additional context variables as relevant when needed. The resulting skill library is thus more robust to irrelevant changes in the environment, versatile across different contexts, and efficient to learn.

Supplemental materials, including additional robot experiments, are accessible here: https://sites.google.com/view/scale-causal-learn-robot-skill
2. Related Work

2.1. Robot Skill Learning

Building robots that can solve a wide variety of complex tasks is one of the fundamental problems in robotics. A popular approach is to learn skills parameterized by the task parameters as these can generalize over related tasks. Prior works (Da Silva et al. (2012, 2014)) show such parameterized skills lie on a low dimensional piecewise-smooth manifold in the context space and identify this structure using ISOMAP (Tenenbaum et al. (2000)).

For higher dimensional problems, it becomes infeasible to learn directly in the full context space. (Konidaris and Barto (2007)) propose learning skills in agent-space which is generated by a feature set that retains the same semantics across a range of problems. Another approach is to learn a library of simple parameterized skills which can be composed to solve more complex tasks (Kaelbling and Lozano-Pérez (2017); Wang et al. (2021b)). (Peters et al. (2013)) use this idea to have a robot play table tennis by learning and combining multiple motor primitives. (Pahić et al. (2021)) address the high-dimensionality of the problem by learning a latent space for skills using an autoencoder.

2.2. Intuitive Physics

Intuitive physics is the ability to approximately predict and model the physical world without explicit understanding of the underlying dynamics (Kubricht et al. (2017)). Literature in cognitive psychology has suggested that humans develop mental intuitive physics models to support fast prediction and understanding of complex physical scenes which enables physical reasoning (Battaglia et al. (2013)). Computational learning of intuitive physics have been successful, enabling reinforcement learning and planning applications owing to the models ability for forward prediction (Ha and Schmidhuber (2018); Hafner et al. (2020); Wang et al. (2021a)).

Recently, interest has garnered in internal models that enable interventions upon the scene variables (Ahmed et al. (2020)), enabling causal reasoning to provide dimensionality reduction and zero-shot transfer (Lee et al. (2021)). In our work, our causal inference engine can be viewed as an internal model that supports causal reasoning through interventions to elicit the physical mechanisms by which the data arises.

3. Preliminaries

3.1. Skills

We model each robot skill $K$ of the robot using the options framework (Sutton et al. (1999); Konidaris and Barto (2009)). An option consists of three components: (a) a control policy $\pi(s)$, (b) an initiation, or precondition, set $I$ which defines the states in which the option can be executed and (c) a termination condition $\beta(s)$ which defines the states in which the option must terminate. In addition, we use a probabilistic notion of skill preconditions (Konidaris et al. (2015)): the skill precondition $Pre(s)$ is the probability that it can be successfully executed at a given a state $s$.

3.2. Policy Learning

The robot will be learning a set of skills $K = \{K_1, \ldots, K_K\}$ it can execute using a given policy $\pi_\theta$, which is a low-level controller parameterized by policy parameters, $\theta$. An example of such a controller is a low-level policy that directly involves robot actuation, e.g., sending torque commands,
which can be specified via a trajectory. The learned skills are solutions to a family of manipulation tasks. Each manipulation task is modeled as a manipulation MDP (Kroemer et al. (2021)):

\[ M = (S, A, R, T, \gamma, \tau) \]  

where \( S \in S \) is the state space, \( A \in A \) is the action space, \( R \) is the reward function, \( T \) is the transition function, \( \gamma \) is the discount factor, and \( \tau \) is additional task information.

Note that because the skill policy is an option, we are learning an upper-level policy, where the policy \( \pi_w \) regresses to \( \theta \). Therefore, for our learning problem, we are solving a one-step MDP over context variables \( C \), which are represented as context \( C \in C \), and parameter variables \( \Theta \), which is represented as \( \theta \in P \), where \( P \) is the space of the low-level policy \( \pi_\theta \). We define the context as a set of variables the robot should conduct causal reasoning over in order to predict the parameters for \( \pi_\theta \) to generalize over \( C \).\(^1\) In practice, we typically define the context \( C \) as the initial state \( S_0 \) plus the additional task information, \( \tau \), e.g., \( S \times \tau \). Importantly, not all variables in \( C \), i.e., dimensions of \( C \), are necessary for the task.

### 3.3. Simulation as a Causal Inference Engine

Our approach assumes the robot has access to a causal inference engine that has two key components: 1) the ability to conduct interventions and 2) forward simulates system dynamics. Both of the components enable causal learning in our work. As described in Fig. 2, the simulator model, \( W := (C_S, C_T) \), is formalized as two SCMs:

- a scene SCM \( C_S \), that instantiates the scene given a scene context vector, \( c \in C \)
- a transition SCM \( C_T \) that approximates the domain forward dynamics, \( T \), as the robot interacts with the world through \( \theta \) starting from scene initialized from \( C_S \).

These two components capture the spatial structure inherent to the scene itself (\( C_S \)), as mentioned by (Schölkopf et al. (2021)), and the spatiotemporal structure of the robot interacting with the world (\( C_T \)). The scene SCM \( C_S \) is defined by structural equation models with scene variables \( \Psi \). Of these scene variables, the robot only conducts causal reasoning for policy learning with respect to \( C \subseteq \Psi \), which are instantiated from \( c \) via interventions, i.e., \( C_i := c_i \). We only consider interventions that would yield a steady-state solution and are physically realizable, which excludes invalid scenes that are could be simulated but are not possible in reality (e.g., object penetration). The variables \( \Psi \setminus C \) represent variables that the simulator requires but the robot is not generalizing the policy over, e.g., gravity. We assume the representation of \( C \) is disentangled. How the robot chooses \( C \) from \( \Psi \) is left for future work, since the set of all possible scene variable \( \Psi \) could be large. Such a choice may be learned directly causal representation learning (Schölkopf et al. (2021)) or be provided directly to the robot. Non-learned representations can be useful since certain robot functionalities already work in disentangled spaces. For example, an object pose estimator may give outputs in \( SE(3) \). Furthermore, this representation can also be human-specified via human-robot teaching (Lee and Kim (2013)). This is because we are learning the structure for a high-level policy; the low-level (time-varying) policy is not examined in this work. Lastly the robot can only interact with the simulator via black-box interaction: specifying \( c \) and \( \theta \) as input, and observing the arising states

---

\(^1\) In contextual MDPs, \( \tau \) is referred to as the context vector. However, in our work, we refer to context as any information the robot might want to know about, so we refer to \( \tau \) as the task information to avoid confusion.
and reward $R$. The robot does not intrinsically know the structure of $\mathcal{W}$, so for this work, we specify it manually, but in principle it could be learned via a world models approach that admits causal interventions.

### 3.4. Regional Causal Skill

We formalize each robot skill $\mathcal{K}$ as a **Regional Causal Skill** (RCS), which extends previous work in precondition skill learning by interleaving causal structure, where $\mathcal{K} := (\pi, D, \text{Pre}, \text{Eff})$:

- $\pi$ is the control policy,
- $D$ is the *data generating region* (DGR),
- $\text{Pre}$ is the *precondition*,
- $\text{Eff}$ are the outcome *effects*.

Intuitively, a RCS can be viewed as arising from an underlying data generating process: under the region $D$ the data $\text{Eff}$ arises from the process of the robot executing skill via control process $\pi$. We assume the policy function class is given (e.g., linear policy, non-linear policy). This is specifically the policy $\pi$ that will be learned later that generalizes over particular context variables. The preconditions $\text{Pre}$ of a skill refer to the space of variables $P$ the robot skill $\mathcal{K}$ can be executed to
yield effects Eff. In this work, we model this as a probabilistic classifier that predicts the probability of successful execution of a skill from a given context. However, for our work, our skills will only execute open-loop with fixed-duration, so termination conditions not evaluated in this work. For our effects, we are only considering the effect of Eff. More rich symbolic effects could be represented in this framework; we leave this for future work.

3.5. Relevant Variables for Skills

Unique to our work, we are not only learning the policy and preconditions of a skill – we are learning the structure itself. Specifically, we identify the variables that are required to fully characterize the skill, and learn functions using these variables as inputs.

We assume the robot is given a context space \( C \) under which multiple skills will be learned. The context space \( C \) is a bounded, disentangled space where each dimension has a given lower and upper bound. The set of context variables \( C \) then is the representation of the context space \( C \) used for causal reasoning. No restrictions are given on the context space \( C \), although we assume that the true relevant variables to generalize via skills are present such that the causal system is identifiable. Because our skills are open-loop with respect to \( \theta \), in practice, our context space typically comprises the initial value of state space variables (variables expected to be time-varying) and constants, which are any variable that is not a state space variable.

Importantly, the minimum context variables needed to characterize \( K \) may be less that the total number, \( C \). Intuitively speaking, out of a possible set causes or properties, the robot should use causal reasoning to determine a reduced set of variables. This step can occur after a high-level heuristic, such as only considering contexts that could possible apply with approaches such as meta-level priors (Kroemer and Sukhatme (2016)), rather than an unbounded list of all possible causes.

A context variable \( C \) is considered policy-relevant if it exists as a part of the policy model, i.e. \( C \subseteq A \). Similarly, a DGR-relevant context variable lies within \( D \), i.e. \( C \subseteq D \). For our work, we only consider fixed-time rollouts, and thus the termination-relevant variables are always \( \emptyset \). Therefore, the entire set of variables needed to specify skill-relevant variables \( A \cup D \) is the union of underlying relevant variables of the skill components.

3.6. Problem Statement

The goal of this work in structure learning for robot control is as follows: Using self-supervised learning within causal inference engine \( W \), learn the structure and policy of Regional Causal Skills \( K \) using variables \( A \cup D \) that may arise through controller \( \pi_{\theta} \) over context space \( C \). The structure of a RCS is learned by conducting interventions to determine policy-relevant and DGR-relevant variables \( A \) and \( D \), respectively. Then, the precondition region \( \mathcal{P} \) is learned over \( C \), and the policy \( \pi_{w} \) is learned over \( C \).

4. Skill Structure Learning in Simulation

The first step in the structural sim-to-real pipeline is to use simulator \( W \) to learn skills through self-supervision. We describe this process below in Alg. 1
**Algorithm 1** SCALE: SKILLS FROM CAUSAL LEARNING

**Input:** causal inference engine $W$, context space $C$, controller $\pi_\theta$, reward solved threshold $R_S$, number of samples $n$, skill policy function class $f_\pi$, number of evaluations $m$

**Initialize:** batch dataset $D \leftarrow \emptyset$, skills $K \leftarrow \emptyset$

// Collect training data
for $i = 1$ to $n$ do
  $c \leftarrow \text{SAMPLEVALIDSCENE}(W, C)$
  $(\theta, R) \leftarrow \text{TRYTOSOLVE}(W, c, \pi_\theta)$
  $\text{TaskSolved} \leftarrow R > R_S$
  if $\text{TaskSolved}$ then
    $(A, D) \leftarrow \text{LEARNRELEVANTVARIABLES}(W, c, \pi_\theta, \theta, R)$
    $D \leftarrow (A, D, c, \theta, R)$
  end
end

// Separate into k datasets, one for each skill
$(D_1, \ldots, D_k) \leftarrow \text{SPLITINTOSKILLDATASETS}(D)$

// Train skills
for $j = 1$ to $k$ do
  $(A, D, C_X, \theta Y) \leftarrow D_j$
  // Train DGR
  $P_X \leftarrow \text{REDUCEDIMS}(C_X, D)$
  $D \leftarrow \text{TRAINDGR}(P_X)$
  // Train Policy
  $A_X \leftarrow \text{REDUCEDIMS}(C_X, A)$
  $(A^+_X, \theta Y^+) \leftarrow \text{KEEPDGRINLIERS}(D, A_X, P_X, \theta Y)$
  $\pi \leftarrow \text{LEARNPOLICY}(f_\pi, A^+_X, \theta Y^+)$
  // Train Preconditions
  $(C_{xe}, R_{ye}) \leftarrow \text{EVALUATEPOLICY}(W, C, \pi, m)$
  $\text{Pre} \leftarrow \text{TRAINPRECONDITION}(C_{xe}, R_{ye}, R_S)$
  // Set Effects
  $\text{Eff} \leftarrow \text{TaskSolved}(\text{true})$
  // Compose Skill
  $K \leftarrow (\pi, D, \text{Pre}, \text{Eff})$
end

Result: learned skills $K$
4.1. Batch Data Collection

In the first part of SCALE, the robot interacts with the simulator \( \mathcal{W} \) to collect skill training data into batch dataset \( D \). The robot samples random scenes encoded by \( c^i \) through \textsc{SampleValidScene}. Then, the robot attempts to solve the task in \textsc{SampleValidScene}. In principle, any model-free policy learning algorithm would suffice. We use Relative Entropy Policy Search (REPS) (Peters et al. (2010)) because the dimensionality of the policy is sufficiently limited to permit policy search solutions. \textsc{LearnRelevantVariables} is modified from CREST which is described in (Lee et al. (2021)). This modification extends CREST to determine both locally and globally relevant variables. The statistical check used in CREST here suffices to do a simple change, but in principle more sophisticated dependency metrics could be used, e.g. Hilbert-Schmidt Independence Criterion (HSIC) (Gretton et al. (2008)). We note that to properly characterize the dependencies between context variables and policy parameters, solutions that yielded local optima were most useful.

4.2. Splitting Batch Data into Skill Data

At this stage, we have a large dataset and we wish to find the \( k \) SCMs that fully characterize the data, and in what region each SCM applies. For our work, we assume that the splitting of \( D \) into \( k \) skill datasets can be done by assigning unique sets \( A \) into the same dataset \( D_j \). This assumption will not hold in general, but is sufficient for the tasks we examine in this work.

4.3. Skill Training

The last stage of the algorithm trains skill \( K_j \) using \( D_j \). This process involves using \textsc{ReduceDims} to reduce datasets by dimensions according to the corresponding variable. We train the DGRs using a one-class SVM, and use the resulting inliers as training data for the policy. \textsc{LearnPolicy} for our work is learning a regression model, but in principle could be used for RL as in (Lee et al. (2021)). Then, the policy is evaluated over the context space to determine where the task is solved, allowing a precondition classifier to be trained to predict probability of skill success. For our work, we use a nonlinear SVM classifier that has probability estimates.

5. Experimental Results

We conduct skill learning experiments with SCALE for two manipulation tasks with the Franka Emika Panda robot: block stacking and peg insertion (Fig. 3). Both tasks are chosen for being emblematic of manipulation tasks in industrial settings. High-precision control is particularly desirable in some industrial applications (Luo et al. (2019)), motivating the ability to include only relevant variables for a policy by excluding spurious features that could degrade performance.

We conduct our experiments in NVIDIA IsaacGym (Liang et al. (2018); Makoviychuk et al. (2021)), a high-fidelity physics simulator that also serves as our causal reasoning engine \( \mathcal{W} \). We implement a custom library that implements the scene SCM \( C_S \) to facilitate scene creation and interventions. The forward simulation of physics provides \( C_T \).

**Task representation.** In the block stacking task, the robot starts with a source block \((B_1)\) grasped, and it learn to place it on top of a target block \((B_2)\). To do this, the robot uses a controller \( \pi_{\theta} \) that defines the trajectory for the robot end-effector to traverse via impedance control. This trajectory is parameterized by \( \theta_b = [\theta_{\Delta x}, \theta_{\Delta y}, \theta_{\Delta z_a}, \theta_{\Delta z_d}]^T \in \mathbb{R}^4 \), which specify waypoints the
Figure 3: We learn regional causal skills using the Franka Emika Panda robot for two manipulation tasks: 3(a) block stacking and 3(b) blocks insertion. For both tasks, skills are learned using causal inference in simulation.

robot follows sequentially. Specifically, these parameters characterize a trajectory where the robot lifts the source block vertically, moves horizontally, and descends vertically and ungraps the block.

For this task, the context variables $C_B = \{C_{B_1}, \ldots, C_{B_{NB}}\}$, which is the union of context variables for each of $N_B = 5$ blocks. The context variables for each block $b$ are $\{x^w_b, y^w_b, \psi_b, h_b, R_b, G_b, B_b\}$, yielding a 35-dimensional context space for this problem. Here, $x^w_b$ and $y^w_b$ are the world $x$- and $y$-positions of the block, and the block’s orientation is represented by a rotation angle $\psi_b$ around the block’s vertical axis ($z$). The $z$-dimension (height) of the block is $h_i$. The color of the block is specified as a red-green-blue tuple, $\{R_b, G_b, B_b\}$. Note that the block vertical position $z^w_b \in \Psi$ is not part of the context, as we only consider cases where the scene can be initialized into a steady state condition. Thus, $z^w_b := \frac{1}{2} h^b + N_{z^w_b}$, where $N_{z^w_b}$ arises from the height of the plane on which the blocks are stacked.

The reward function for the task is $R = R_B - \alpha_L L - \alpha_e e - \alpha_d d$, where $R_B = 10$ is a bonus term obtained when the block is successfully stacked, $L$ is the total end-effector path of the robot ($\alpha_L = 1$), $e$ is the L2 norm error between the source block at the time of release and the goal ($\alpha_e = 1$), and $d$ is the distance the source block travels between the point it was ungrasped to its final position ($\alpha_d = 1$).

Skill Learning Results on Simple Experiments. In order to first build intuition for skill learning, we first provide two simple examples in Appx. A. These examples use two-dimensional context spaces to facilitate visualization of the SCALE skill learning process.

Skill Learning Results. In the block stacking domain, SCALE learned skills enumerated in Table 1. Skills $K_1$ and $K_5$ are the most successful skills at completing the task; both require significantly fewer parameters than the size of the context space. With the skill selection percentage indicated, this family of skills solved the task 95.22% (299/314) times. Interestingly, skills $K_6 - K_9$ correspond to skills arising from scenes where the robot encountered this variable. Note that its
Table 1: Skill learning results for the block stacking task using linear policies. The quantity of tries for Solve Rate and Skill % was 314, except for \( K_{11} \), which was 313. The robot primarily used \( K_1 \) and \( K_5 \), which arose most frequently when the robot could move the block without obstruction from another block. Other skills arise when lower frequency under conditions where the robot policy accounted for these obstructions. Note that \( K_{11} \) was a result of policies that had relatively greater stochasticity, so all variables were considered relevant in this case. Skills \( K_7 \) and \( K_{11} \) could not solve the task, so it was excluded from consideration for skills for the robot to select in task evaluation.

<table>
<thead>
<tr>
<th>Skill</th>
<th>( x_1, y_1, x_2, y_2 )</th>
<th>( x_1, y_1, \psi_1, x_2, y_2 ), ( h_2 )</th>
<th>( x_1, y_1, \psi_1, x_2, y_2, h_2 )</th>
<th>( x_1, y_1, \psi_1, h_1, x_2, y_2, h_2 )</th>
<th>( x_1, y_1, \psi_1, h_1, x_2, y_2, h_2, h_3 )</th>
<th>Samples</th>
<th>Solve Rate</th>
<th>Skill %</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K_1 )</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>85</td>
<td>221 (70.38%)</td>
<td>188 (59.87%)</td>
</tr>
<tr>
<td>( K_2 )</td>
<td>{ y_1, y_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>10</td>
<td>16 (5.10%)</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>( K_3 )</td>
<td>{ x_1, y_1, y_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>17</td>
<td>20 (6.37%)</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>( K_4 )</td>
<td>{ x_1, y_1, x_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>3</td>
<td>1 (0.32%)</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>( K_5 )</td>
<td>{ x_1, y_1, x_2, y_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>294</td>
<td>283 (90.13%)</td>
<td>125 (39.81%)</td>
</tr>
<tr>
<td>( K_6 )</td>
<td>{ x_1, y_1, x_2, y_2 }</td>
<td>{ x_1, y_1, x_2, y_2, h_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>3</td>
<td>1 (0.32%)</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>( K_7 )</td>
<td>{ x_1, y_1, x_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>4</td>
<td>0 (0.00%)</td>
<td>–</td>
</tr>
<tr>
<td>( K_8 )</td>
<td>{ x_1, y_1, x_1, x_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>3</td>
<td>1 (0.32%)</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>( K_9 )</td>
<td>{ x_1, y_1, \psi_1, x_2, y_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>12</td>
<td>66 (21.02%)</td>
<td>1 (0.32%)</td>
</tr>
<tr>
<td>( K_{10} )</td>
<td>{ x_1, y_1, \psi_1, h_1, x_2, y_2 }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>{ }</td>
<td>3</td>
<td>1 (0.32%)</td>
<td>0 (0.00%)</td>
</tr>
<tr>
<td>( K_{11} )</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>all</td>
<td>5</td>
<td>0 (0.00%)</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 2: Comparison task completion using SCALE skill learning versus a monolithic policy.

<table>
<thead>
<tr>
<th>Policy Function Class, ( f_\pi )</th>
<th>Approach</th>
<th>Task Solved Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>SCALE (ours)</td>
<td>95.22% (299/314)</td>
</tr>
<tr>
<td>Nonlinear</td>
<td>SCALE (ours)</td>
<td>89.81% (282/314)</td>
</tr>
<tr>
<td>Linear</td>
<td>no skill</td>
<td>82.48% (259/314)</td>
</tr>
<tr>
<td>Nonlinear</td>
<td>no skill</td>
<td>58.92% (185/314)</td>
</tr>
</tbody>
</table>

relatively sparse samples suggests that random scenes alone would be insufficient to collect enough data to characterize these policies, implying future direction towards active learning of skill DGRs.

**Comparison to Full Context Space Policies.** To highlight the utility of learning compact, minimally parameterized skills, we conduct an additional experiment where we compare skill learning to an approach that only learns one monolithic policy for the entire context space. For this experiment, instead of separately training different skills, we instead train one policy using this data as input.

Table 2 shows our results for two classes of policy model (linear, nonlinear). Results for nonlinear functional classes are shown with a multilayer perception with 1 hidden layers of 16 neurons each (Pedregosa et al. (2011)). We also compared increasing the number of hidden layers to 2 and 3, but there was little change to performance.

Our results suggest that our skill-centric approach provides greater task completion rate for both policy function classes. Each of our skills capture one DGR, whereas the no skill approaches attempt to fit a model to all DGRs. The performance of the no skill linear policy suggests that this domain is weakly linear. The performance of the no skill nonlinear policy is notable and could be attributed to fewer samples (439) than what would otherwise be typically available for neural networks.
5.1. Peg-in-Hole Insertion

Our second domain is peg-in-hole insertion under sensing uncertainty. It requires the robot to insert a cuboidal peg of cross-section $1cm \times 1cm$ into a cuboidal hole of cross-section $1.3cm \times 1.3cm$. While this task is easy if the location of the hole is accurately known, practically there is always some uncertainty in the pose of the hole due to sensing error. We mimic this in simulation by adding noise sampled from a Gaussian distribution $\mathcal{N}(0, 0.3^2cm^2)$ to the location of the hole every time the robot attempts it. Due to this uncertainty, a naive strategy of directly trying to push the peg down at the given location of the hole achieves a success rate of only $34\%$. To address this, (Brost (1986)) propose an appealing approach in which the robot takes uncertainty reducing actions by taking advantage of contact with the environment (for example a fixture next to the hole). Our goal in this experiment is to learn such skills in a Physics simulator.

Task Representation Each assembly task has 4 axis-aligned cuboidal fixtures of fixed dimensions around the hole. The robot gets an 8-dimensional context vector $c = [x_1, y_1, \ldots, x_4, y_4]$ which contains the $(x, y)$ coordinates of these fixtures with respect to the hole. The positions are different in every task but it is always possible for the robot to localize against any of the walls to complete the task. We use a 6-parameter policy space. Each policy consists of three $(\Delta x, \Delta y, \Delta z)$ actions executed in sequence in the robot’s end-effector frame. For every action, $\Delta z$ is hard-coded while $\Delta x$ and $\Delta y$ are parameters that are learnt using RL. Our reward function consists of two terms (1) a penalty based on the Euclidean distance of the peg from the hole (1) a bonus of 10 for successful insertion. We also add a regularization term based on the norm of the policy parameters.

Skill Learning Results The skills discovered by SCALE are enumerated in Table 3. Skills 1-4 each localize against one of the 4 walls around the hole. All of them have high success rates which shows the effectiveness of localization using contact. Skill $K_5$ is interesting as it seems to imply that it is able to solve the task without the help of any wall. However, we observed that the samples that are so classified actually localize against 2 walls instead of just 1. Hence, when we intervene on any one of the two walls, the skill is still able to complete the assembly by taking advantage of the other wall. In other words, our assumption that the context space is disentangled does not hold in this case which leads to this erroneous relevant variable set. Finally, we train preconditions for skills $K_{1-4}$ and evaluate them together on 100 tasks by choosing the best skill for each task. We observe a high success rate of $97\%$ which is not surprising given that all the skills are individually quite robust.

6. Conclusion

We present SCALE, an approach for causal learning of compact, diverse robot manipulation skills from causal interventions in simulation. These skills arise from the skill DGR – an underlying data generating process for how the robot should select parameters to solve a task in a particular context. We assume the robot has access to a causal inference engine, which is an area of future work. Additionally, we will investigate active learning of DGRs to explore using SCALE for tasks defined with higher dimensional spaces than those examined in this work.
Table 3: Skill learning results for the peg-in-hole insertion task.

<table>
<thead>
<tr>
<th>Skill</th>
<th>A</th>
<th>D</th>
<th>#samples</th>
<th>Solve rate (%)</th>
<th>Skill %</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathcal{K}_1)</td>
<td>({x_4})</td>
<td>({x_4})</td>
<td>28</td>
<td>96</td>
<td>18</td>
</tr>
<tr>
<td>(\mathcal{K}_2)</td>
<td>({y_3})</td>
<td>({y_3})</td>
<td>27</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td>(\mathcal{K}_3)</td>
<td>({y_2})</td>
<td>({y_2})</td>
<td>32</td>
<td>98</td>
<td>0</td>
</tr>
<tr>
<td>(\mathcal{K}_4)</td>
<td>({x_1})</td>
<td>({x_1})</td>
<td>26</td>
<td>98</td>
<td>78</td>
</tr>
<tr>
<td>(\mathcal{K}_5)</td>
<td>({})</td>
<td>({x_1, y_2, y_3, x_4})</td>
<td>45</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4: Our approach identifies 5 distinct skills, each with a distinct set of relevant variables. Skills \(\mathcal{K}_{1-4}\) each localize against a different wall which shows up as the relevant variable in the corresponding data generation region. Note that for localization, only distance from the hole along the normal of the wall is relevant. For example, if the wall normal is along the \(x\) axis, then the \(y\) coordinate of the wall’s position is irrelevant. Our algorithm is able to figure this out purely based on interventional queries to the simulator. Skill \(\mathcal{K}_5\) shows up as having no relevant variables. However, we observed that it in fact uses multiple walls for localization. This breaks our assumption that the context space is disentangled and points to a limitation of our work.

References


Appendix A. Block Stacking Intuitive Examples

We present two intuitive examples, that can be easily visualized, in the block stacking domain: \textit{Height-Height} experiment and \textit{Height-Position} experiment. The context space of both the experiments is 2.

A.1. Height-Height experiments

The \textit{Height-Height} experiments include a setup of a source block, a target block, and an obstructing block between the source and target block (fig.4). The task is to place the source block on the target block. The 2 context variables that can be intervened upon are the height of the obstructing block and the height of the target block. The policy for this experiment is parameterized as follows:

1. Policy Parameter 1: how many meters the source block is moved in x-axis once picked up
2. Policy Parameter 2: how many meters the source block is moved in y-axis once picked up
3. Policy Parameter 3: how many meters the source block is moved in z-axis when picking up
4. Policy Parameter 4: how many meters the source block is moved down in z-axis when putting down

The policy behaves as follows:

1. Pick up the source block up in the z-axis according to policy parameter 3
2. Move it in the x-y plane according policy parameters 1 and 2
3. Put the source block down on the target block on the z-axis according to policy parameter 4

In fig.5 we can see the data generating regions, from which 2 skills were identified and their precondition boundary was learned fig.6. The precondition boundaries for each skill is made by applying the skill on all the data samples and scoring them on probability of success, with samples that are more than 50 percent likely for the skill to succeed on to be positive and the other to be negative. From these precondition boundaries, a decision boundary for choosing skills can be constructed by choosing the skill which provides the highest probability of success at each point as the chosen skill as shown in fig.7.

In fig.8 we see the variation in the policy parameters found by REPS over the context space. For policy parameters 1 and 2, the change in context variables has minimal impact on the policy parameters since the x-y translation that needs to be done to move the source block onto the target is the same regardless of the height of the other 2 blocks. For policy parameter 3, it can be seen that what determines the height to pick the source block up seems to be the maximum between the heights of the obstructing block and target block. This makes intuitive sense since the source block will need to go above the tallest block to achieve its task. The connection between DGRs and the context space is also illustrated by this plot as places with high gradients form a DGR since a high gradient shows that a variable is important in a certain region. For policy parameter 4, only when the height of the obstructing block is higher than the height of the target block does how much the block needs to be dropped increase. This result makes sense, because if the obstructing block is low, then the policy is optimized by not picking up the source block too high initially. Thus, the distance it needs to be dropped off is minimized.
Figure 4: Experimental setup for *Height-Height* experiments

Figure 5: Data generating regions identified after collecting data samples in *Height-Height* experiment
Figure 6: Precondition boundary learned for skill 1 (a) and skill 2 (b) in *height-height* experiment. Color of the region represents probability of success. Blue points are classified positive and red points are negative. Diamond points are from the test set and circle points are from training data.

Figure 7: Decision boundary learned for skill selection in *Height-Height* experiment.
Figure 8: Policy parameter variations in *Height-Height* experiment. Y-axis is height of obstructing block in meters. X-axis is height of target block in meters. Color of graph represents the value of the parameter according to given range.
A.2. Height-Position experiments

The setup is the same as the Height-Height experiments, except for the context variables which are the height of the obstructing block and the x-axis position of the block (fig. 9).

In fig. 10 we can see the data generating regions as well as a distinctive gap between the 2 data-generating regions occurring as columns on the x-axis. This happens because there are no valid states for the environment to be created with those obstructing block positions as it would then be intersecting with the other blocks. The data-generating region for skill 1 has no relevant context variables since the position of the obstructing block does not affect the task in any way in those regions.

In fig. 13 we see the variation in the policy parameters found by REPS over the context space. For policy parameters 1, 2 and 4, the change in context variables has minimal impact on the policy parameters as the position of the obstructing block under any context doesn’t change these parts of the policy. For policy parameter 3, changes in the height of the obstructing block, if the x position of the block is central, can lead to a large change in this parameter of the policy. This happens when the obstructing block is central and high, and the robot must carry the source block over the obstructing block, which means it must increase how high it must lift the block up initially along the z-axis. The graph also shows evidence for a data generating region here as there is a high change in gradient when changing the height of the obstructing block, which suggests that height is a relevant context variable in that region.

Appendix B. NVIDIA Isaac Gym

Our experiments were performed using NVIDIA’s Isaac Gym simulation environment, as shown in fig. 14.
Figure 10: Data generating regions identified after collecting data samples in *Height-Position* experiment

Figure 11: Precondition boundary learned for skill 1 (a) and skill 2 (b) in *Height-Position* experiment. Color of the region represents probability of success. Blue points are classified positive and red points are negative. Diamond points are from the test set and circle points are from training data.
Figure 12: Decision boundary learned for skill selection in \textit{Height-Position} experiment
Figure 13: Policy parameter variations in Height-Position experiment. Y-axis is height of obstructing block in meters. X-axis is the position of the target block on the x-axis in the simulator in meters. Color of graph represents the value of the parameter according to given range.
Figure 14: We use NVIDIA Isaac Gym to run many simulations in parallel. This figure shows 32 parallel environments running the block stacking manipulation task.