Learning Multi-step Manipulation using Symmetry-aware Reinforcement Learning

Hai Nguyen[1], Huitan Mao[1], Manikantan Nambi[1]

***Abstract*— While many existing grasping models can be highly reliable in picking objects in most cases, challenging scenarios persist in industrial automation where objects are difficult to grasp—such as when positioned in corners, occluded by other items, or tightly clustered. These challenges are prevalent in smart manufacturing and logistics systems, where current robotic systems often require costly human intervention for handling difficult cases. To address this automation gap, we propose a multi-step manipulation approach that combines non-prehensile and prehensile actions through two collaborative policies: a main policy responsible for picking, and a supporting policy that assists by rearranging the scene when the main policy lacks confidence. In particular, we leverage equivariant neural networks to encode SE(2) symmetries in both policies, enabling sample-efficient reinforcement learning from visual observations. Using a symmetry-aware DQN-based approach, we first train the main policy, then train the supporting policy to maximize the main policy's confidence through a value-based reward function. Our method significantly outperforms single-policy approaches, achieving 1.3-2$\times$ better performance across five manipulation tasks in simulation. Crucially, we demonstrate efficient on-robot learning directly on hardware (1-2 hours) for three tasks, showcasing the practical viability of our approach for smart industry applications where rapid deployment and adaptation are essential.**

I. INTRODUCTION

Despite the remarkable success of many powerful grasping models [1]–[5] in relatively structured and uncluttered settings, challenging real-world scenarios still exist where safe and effective grasp poses are difficult or even impossible to predict. Prominent examples include picking heavily occluded objects (completely hidden behind others or tightly clustered together in bins) and corner-positioned objects in confined workspaces where collision-free grasps are extremely hard to find due to geometric constraints imposed by surrounding walls or neighboring items. This limitation has significant practical implications: in warehouse fulfillment centers and manufacturing production lines, current robotic systems often fail on these difficult scenarios, necessitating costly, time-consuming, and operationally disruptive human intervention. Fortunately, additional preparatory actions—such as pushes, drags, or intermediate picks—can strategically rearrange the scene and transform these initially ungraspable configurations into graspable ones. This observation naturally motivates multi-step manipulation, where non-prehensile and prehensile actions are sequentially combined to enable successful grasping, ultimately automating cases that currently require manual human handling.

[1]Amazon Robotics, North Reading, MA, USA. {hainhbk, mhuitan, mnambi}@amazon.com. Project site: https://sites.google.com/view/equi-multi-step.

In this work, we propose a two-policy framework for multi-step manipulation built upon visual DQN [6]. Both policies leverage equivariant neural networks [7] that encode SE(2) symmetries (translation and rotation) directly into the policy architecture, substantially improving sample efficiency by reducing the effective action space and enhancing generalization across diverse scene configurations. Within this framework, a *main* policy is responsible for task completion and operates confidently in the majority of scenarios. When its confidence drops below a predefined threshold—indicating a potentially ungraspable scene, a *supporting* policy is triggered to intervene and modify the workspace layout, for example by rearranging cluttered objects or dragging corner-trapped items toward open, more accessible areas. Training follows a two-stage curriculum: the main policy is trained first using sparse binary rewards ($+1$ for each successful pick), after which it is frozen and used to supervise the training of the supporting policy. Specifically, the supporting policy is optimized to maximize the main policy's post-intervention state value through a value-based reward signal, thereby learning preparatory actions that directly improve downstream grasp success.

Our main contributions are as follows: (1) A sequential two-stage training framework for multi-step manipulation that combines a main picking policy with a supporting rearrangement policy trained using reinforcement learning (RL) via a value-based reward signal—demonstrating a practical approach to automating challenging scenarios in smart manufacturing systems. (2) Integration of equivariant neural networks encoding SE(2) symmetries directly into both policy architectures, achieving superior sample efficiency compared to conventional data augmentation approaches—enabling the policies to generalize across diverse scene configurations without requiring exhaustive training data. (3) Extensive evaluation across five simulated and three real-world manufacturing scenarios, demonstrating 1.3–2$\times$ improvement of dual-policy systems over single-policy baselines, and rapid on-robot learning within just 1–2 hours of robot interaction—validating the practical viability of the proposed framework for smart industry applications where fast deployment and minimal human supervision are essential.

II. RELATED WORK

A. Learning Multi-step Manipulation

Learning multi-step manipulation policies with RL has been explored extensively, primarily focusing on push-pick synergy for grasping objects in clutter. Many studies [8]–[19] investigate "singulation"—using pushing to separate

closely positioned objects to facilitate picking. However, these singulation actions may be suboptimal since they are not explicitly optimized for subsequent grasping. Moreover, prior work predominantly focuses on box-like objects and learns in simulation with sim-to-real transfer, whereas we address diverse object shapes, explore multiple manipulation synergies (pick-place, drag-pick, push-pick), and enable direct on-robot learning.

Prior approaches span heuristic methods [17], planning [8], [10], [14], [20]–[22], and deep RL [13], [23]–[25]. Among RL methods, VPG [13] and [24] are most related to ours. While VPG trains two policies on-robot, [24] extends VPG but only performs sim-to-real transfer. Our method differs from VPG in several key aspects: (1) We use a pre-trained main policy for stability rather than joint training. (2) We encode symmetries directly via equivariant architectures [26] rather than indirectly through data augmentation. (3) We use value-based rewards, i.e., the change in the main policy state value $V^M(s_{t+1}) - V^M(s_t)$, to explicitly train the supporting policy to create favorable grasp scenarios, whereas VPG uses scene-change rewards for push actions to optimize for the largest scene changes. Unlike [24], which learns pushing from grasp quality without considering symmetry, our symmetry-aware approach is crucial for achieving superior performance and sample efficiency.

B. Symmetry-aware Policy Learning for Robot Manipulation

Exploiting domain symmetries significantly improves sample efficiency, particularly for on-robot learning where data collection is expensive. For top-down observations, symmetry arises because rotating or translating the scene induces corresponding transformations in the optimal action. Similar to how CNNs [27] encode translation invariance, equivariant networks [26], [28] can directly encode these symmetries into RL agents. Pioneering work by Wang et al. [29]–[33] demonstrates the effectiveness of this approach. While prior works only consider a single policy, we leverage symmetry-aware agents from [29] for both policies, enabling efficient on-robot learning within 1-2 hours across all tasks.

III. BACKGROUND

A. Augmented State Representation (ASR)

Augmented State Representation (ASR) [29] factorizes the action space to accelerate visuomotor policy learning in DQN-based RL. Consider a top-down pick action $a = (x, y, \psi)$, where (x, y) specifies the pick location in the pixel coordinate and ψ is the rotation around the Z-axis (see Fig. 1). Instead of learning a complex joint Q-function $Q(s, (x, y, \psi))$, ASR decomposes it into two simpler functions: a spatial Q-function $Q^1(s, (x, y))$ and a rotation Q-function $Q^2(s, \psi | x, y)$, enabling greedy selection of both location and rotation. As illustrated in Fig. 1, given a top-down depth image, Q^1 outputs a pick map with the same spatial dimensions as the input, where each pixel value estimates the picking quality at that location. The optimal pick location (x^*, y^*) corresponds to the maximum value (brightest point) in this map. Given (x^*, y^*), ASR then evaluates Q^2 using a local patch around (x^*, y^*) to select the optimal rotation ψ^*. For top-down grasps, the grasp height z is determined from the depth value at (x^*, y^*) to avoid collisions. For 6-DOF grasps, ASR extends this framework with three additional Q-functions $Q^{3,4,5}$ to select z, ϕ (rotation around X), and θ (rotation around Y) (see [29] for more details). Among our five experimental tasks, four use the planar structure shown in Fig. 1 (top-down grasps only), while one (`Drag-Pick`) employs the 6-DOF extension.

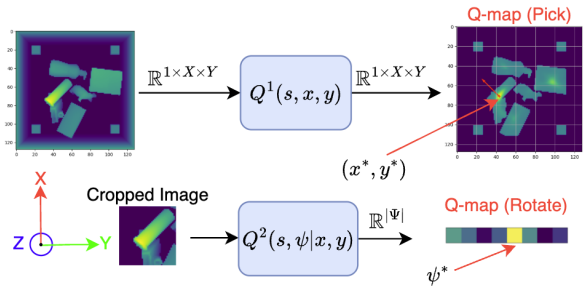


Fig. 1: Illustration of augmented state representation (ASR) [29]. Q^1 selects the pick point (x^*, y^*), and Q^2 selects the Z-rotation ψ^* based on a cropped image around (x^*, y^*).

B. Leveraging Symmetries in Robot Manipulation

Robotic manipulation from visual input inherently exhibits domain symmetries that can be exploited for sample-efficient policy learning. For instance, if we rotate or translate the scene in Fig. 1, the pick Q-map $Q^1(s, (x, y))$ should transform correspondingly—this is the equivariance property of the Q-function under SE(2) transformations [32]. Incorporating such symmetry improves efficiency because the Q-function automatically generalizes to rotated and translated configurations without requiring additional training data. Prior work [30], [32] achieves symmetry-aware Q-functions by first defining a discrete symmetry group to approximate continuous SE(2) symmetry. A common choice is the cyclic group $C_4 = \{0°, 90°, 180°, 270°\} \subset \text{SE}(2)$, which offers a favorable trade-off between expressiveness and computational cost. Given such a group, steerable convolution layers [26], [28] construct equivariant architectures that explicitly encode these symmetries into the network structure. While data augmentation with rotated and translated inputs can also induce symmetry awareness, it does so only indirectly through the training distribution, resulting in slower convergence and higher sample complexity [26], [34]. This work leverages C_4 symmetry to learn multi-step manipulation in simulation and on real hardware via RL—extending prior single-step demonstrations [7], [35], [36].

IV. PROPOSED METHOD

We propose learning two collaborative policies for manipulation tasks (Fig. 2): a main policy that outputs most actions and a supporting policy that intervenes when the main policy lacks confidence. The main policy acts whenever it is confident in achieving the task goal. When confidence

drops below a threshold, the supporting policy is invoked to modify the scene, restoring the main policy's confidence. For example, the supporting policy can drag corner-positioned objects to open areas where the main policy can confidently grasp them without fear of potential collisions.

A. Action and State Representations

Actions consist of motion primitives (*Pick, Push, Place, Drag*) and pre-primitives (*Pre-Push, Pre-Drag*) that position the arm for subsequent *Push* and *Drag* actions. Table I defines the action sequence for each primitive. Specifically, *Pick* and *Place* primitives move the end-effector to the target pose, actuate the gripper (close for pick, open for place), and return to a home position to avoid occluding a top-down camera. *Push* and *Drag* use simplified planar motion, moving only in XY while maintaining constant end-effector orientation and height, then returning home. *Pre-Push* and *Pre-Drag* move to the target pose and remain stationary.

States are top-down depth images augmented with binary flags that enforce primitive sequencing: *Pick* $\rightarrow$ *Place*, *Pre-Drag* $\rightarrow$ *Drag*, and *Pre-Push* $\rightarrow$ *Push*. After *Pick*, a flag indicates whether an object is grasped, enabling *Place* only after successful picks. For *Pre-Drag*, the flag is set upon contact with an object; for *Pre-Push*, it is set after execution. The flags are cleared after *Place*, *Push*, and *Drag*. Since pre-primitives do not return home and occlude the camera, we superimpose the gripper finger positions onto the previous depth image to create an occlusion-free observation (see Fig. 3). This representation not only avoids occlusion but also conveys the outcome of pre-primitives, informing subsequent primitive endpoint selections (e.g., where to end push or drag actions).

Primitive	**Definition of Action Sequences**
Pick	To desired pose $\rightarrow$ Open-Close gripper $\rightarrow$ 🏠
Place	To desired pose $\rightarrow$ Open gripper $\rightarrow$ 🏠
Push, *Drag*	To desired XY position $\rightarrow$ 🏠
Pre-Push, *Pre-Drag*	To desired pose and stay
Executed Primitive	**Components of Resultant State**
Pick	Top-down depth, Grasped an object?
Place, Push, Drag	Top-down depth, 0
Pre-Drag	Prev. obs. w/ super-imposed fingers Made a contact w/ an object to drag?
Pre-Push	Prev. obs. w/ super-imposed fingers, 1

TABLE I: Primitive definitions and the resultant states.

B. Learning Main Policy

We train the main policy as a DQN-based agent using the equivariant architecture from [30] to leverage domain symmetries for sample efficiency. The architecture employs a 16-stride UNet [37] backbone with steerable convolution layers [26] that encode SE(2) equivariance. The main policy's *Pick* primitive is parameterized as (x, y, ψ, p) for planar tasks, where (x, y) specifies the 2D grasp position, ψ is the rotation angle, and p indicates the primitive type. In these tasks, grasp height is automatically determined from

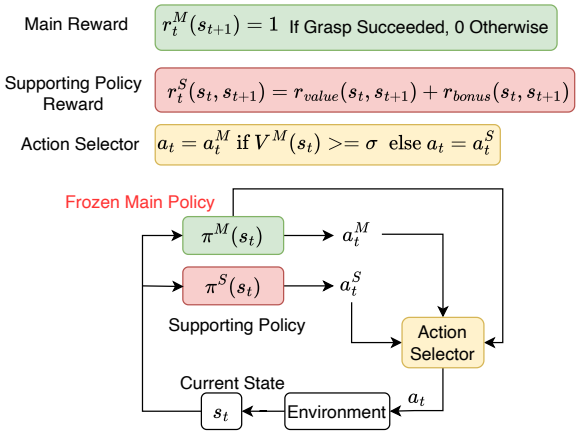


Fig. 2: The main policy is pretrained and receives a binary reward of 1 with a successful pick of an object of interest. The supporting policy is trained with the main policy being frozen, it is rewarded with the change in the main policy's value function and task-dependent bonus rewards.

the heightmap as the maximal z at the selected (x, y) location to avoid collisions. For 6-DOF tasks requiring vertical reasoning (`Drag-Pick`), we use the full parameterization $(x, y, z, \phi, \theta, \psi, p)$ with explicit height z and 3D orientation (ϕ, θ, ψ). The main policy receives a binary reward: $+1$ for successfully picking the target object, 0 otherwise. Following [35], [38], [39], we adapt [30] to a bandit-like setting by terminating episodes immediately after each pick attempt.

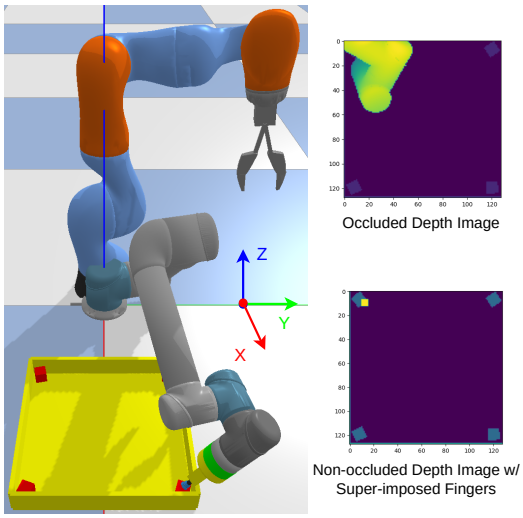


Fig. 3: Simulation setup using PyBullet [40] with Kuka and UR5 robots acting as the main and supporting policies. When the view is obstructed by the robot's pre-primitive actions (e.g., *Pre-Drag*), we superimpose the robot's fingers on the previous observation to create a non-occluded observation and to convey the outcome of the pre-primitive actions.

C. Learning Supporting Policy

The supporting policy uses the same network architecture as the main policy but is trained to maximize the main

policy's confidence. Primitives of the supporting policy fall into two categories based on parameterization. Grasping primitives (*Pick*, *Place*) use the same parameterization as the main policy: (x, y, ψ, p) for 2D position, rotation, and primitive type. Non-grasping primitives (*Push*, *Pre-Push*, *Drag*, *Pre-Drag*) use simplified planar parameterization (x, y, p), controlling only 2D position without rotation while maintaining constant end-effector height during execution.

We quantify the main policy's confidence using its state value $V^M(s)$. Since the Q-function implicitly defines the main policy, we use the value of the greedy policy: $V^M(s) = \max_a Q^M(s, a)$. For example, the main policy's confidence for the state in Fig. 1 corresponds to the maximum value (the brightest point) in the pick Q-map. Given this confidence measure, we define a dense reward function $r_t^S(s_t, a_t, s_{t+1})$ for training the supporting policy:

$$r_t^S(s_t, a_t, s_{t+1}) = r_{\text{value}}(s_t, s_{t+1}) + r_{\text{bonus}}(s_t, s_{t+1}) \quad (1)$$

$$r_{\text{value}}(s_t, s_{t+1}) = V^M(s_{t+1}) - V^M(s_t) \quad (2)$$

$$r_{\text{bonus}}(s_t, s_{t+1}) = \beta\, \mathbf{1}\{\text{success}(s_t, s_{t+1})\} \quad (3)$$

The value-based component r_{value} rewards actions that increase the main policy's confidence by transitioning to higher-value states. The bonus component r_{bonus} provides task-dependent rewards for successful primitive execution, such as grasping an object (for subsequent placement) or making contact with an object (for subsequent dragging). Task-specific reward details are provided in Table II.

Finally, supporting policy episodes span at least two primitives (e.g., *Pre-Drag* $\rightarrow$ *Drag*), with termination occurring after the second primitive completes. Episodes may extend beyond two steps if the pre-primitive fails (e.g., *Pre-Drag* fails to contact any object), allowing retry attempts.

Domain	β	Success Condition
`Block-Pick`	0.05	Pick success
`Block-Push`	0.05	Pick success
`Drawer-Pick`	—	—
`Targeted-Pick`	—	—
`Drag-Pick`	0.05	Contact with object

TABLE II: Bonus reward parameters ($r_{\text{bonus}} = \beta$ when success condition met). — indicates value-based reward only.

D. Action Selection

We employ a simple threshold-based action selector to determine policy control. The main policy acts whenever its confidence (state value) exceeds a threshold $\sigma \in (0, 1)$. When confidence drops below σ, the supporting policy takes control and continues acting until the main policy's confidence recovers above σ. We select $\sigma \in \{0.5, 0.6, 0.7\}$ for each task by analyzing the pre-trained main policy's value distribution during rollouts. Our threshold-based selection is simpler than the Q-value comparison approach used in [13] and [24], which selects whichever policy has higher state value at each step. That strategy can trigger redundant supporting actions—for example, executing an unnecessary push simply because the pushing policy's value exceeds the picking policy's value, even when picking could succeed directly. We use a fixed threshold rather than learning it for three reasons: (1) *Simplicity and interpretability*—the threshold has clear semantics as the minimum confidence for reliable task completion. (2) *Training stability*—a fixed threshold avoids the meta-learning problem of simultaneously learning the supporting policy and when to invoke it. (3) *Avoiding degenerate solutions*—learning the threshold could lead to always preferring one policy. The fixed threshold, calibrated from the converged main policy's value distribution, provides a stable signal throughout supporting policy training.

Domain	σ	Main Policy	Supporting Policy
`Block-Pick`	0.5	*Pick* (JP)	*Pick* (JP)→*Place* (JP)
`Block-Push`	0.5	*Pick* (JP)	*Pre-Push* (JP)→*Push* (JP)
`Drawer-Pick`	0.5	*Pick* (S)	*Pre-Push* (JP)→*Push* (JP)
`Targeted-Pick`	0.7	*Pick* (JP)	*Pre-Push* (JP)→*Push* (JP)
`Drag-Pick`	0.6	*Pick* (JP)	*Pre-Drag* (S)→*Drag* (S)

TABLE III: Task-specific threshold σ to select actions and primitives with gripper types (JP: jaw-parallel, S: suction).

V. LEARNING IN SIMULATION

We conduct simulation experiments using PyBullet [40] across five tabletop multi-step manipulation domains (see Fig. 4) built upon the BulletArm [41] suite. Depending on the task, we employ parallel-jaw (JP) and suction (S) grippers for different primitives (see Table III). All domains use top-down planar actions except `Drag-Pick`, which features 6-DOF actions.

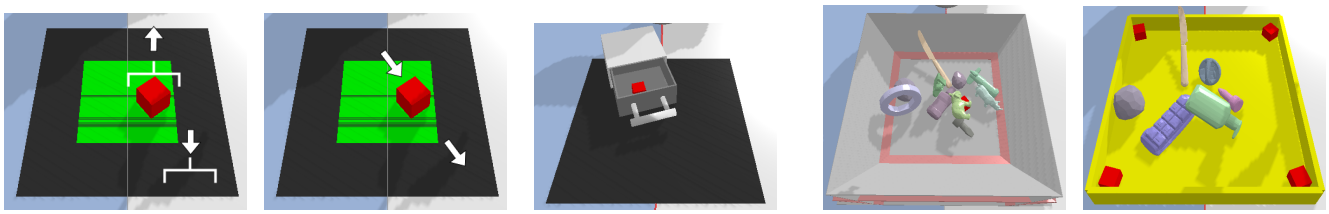
Block-Pick: The main policy must pick a block that can lie outside its reachable workspace (black region in Fig. 4a) 50% of the time. When the block falls in the green region, the supporting policy must pick and relocate it to the black region where the main policy can grasp it. The main policy uses *Pick* actions; the supporting policy uses *Pick* and *Place*.

Block-Push: Similar to `Block-Pick`, but the block when in the green region must be pushed toward the black region. The supporting policy uses *Pre-Push* and *Push* actions.

Drawer-Pick: The agent must open a drawer to pick an object inside. The drawer is initially closed 50% of the time. The supporting policy uses *Pre-Push* and *Push* to open closed drawers, while the main policy uses *Pick* actions.

Targeted-Pick: A target block (red) must be picked. During initialization, it is buried among distractor objects (randomly selected from 86 objects in GraspNet-1B [42]) 50% of the time. This scenario simulates order-based picking in fulfillment systems, where a specific item must be retrieved regardless of surrounding clutter. The supporting policy uses *Pre-Push* and *Push* to separate the target from distractors, enabling reliable picking by the main policy's *Pick* actions.

Drag-Pick: All objects (also from GraspNet-1B) must be picked. While most objects can be directly grasped, four red blocks positioned at corners require dragging to avoid collisions first. This scenario simulates a common industrial challenge in bin clearing, where corner-trapped items are difficult to grasp and often necessitate costly human inter-



(a) `Block-Pick` (b) `Block-Push` (c) `Drawer-Pick` (d) `Targeted-Pick` (e) `Drag-Pick`

Fig. 4: Robot manipulation domains for experiments in simulation. Domains in red are also used real-robot experiments. The poses of all objects are randomized when starting each episode. In `Drag-Pick`, the four red blocks are initialized at the four corners. All domains use top-down actions, except for `Drag-Pick`, which features 6-DoF actions.

Domain	Main Policy	Supporting Policy
`Block-Pick`	$xy\psi$: $128^2 \times 16$	xy: 128^2
`Block-Push`	$xy\psi$: $128^2 \times 16$	xy: 128^2
`Drawer-Pick`	$xy\psi$: $128^2 \times 16$	xy: 128^2
`Targeted-Pick`	$xy\psi$: $128^2 \times 16$	xy: 128^2
`Drag-Pick`	$xyz\phi\theta\psi$: $128^2 \times 36 \times 16^3$	xy: 128^2

TABLE IV: Action space discretization for main and supporting policies across all tasks.

Method	Sequential	Data Aug	Δ Scene Reward
M-S-Equi (Ours)	✓		
M-S-Norm-Aug	✓	✓	
M-S-Equi-Scene	✓		✓
VPG-Equi*			✓
M-Equi			—
M-Norm-Aug	—	✓	—

TABLE V: Baseline comparison. Methods in red use both main and supporting policies. Sequential: supporting policy trained after main policy is pre-trained. M-Equi and M-Norm-Aug are the pre-trained main policies for M-S-Equi and M-S-Norm-Aug, respectively. Data Aug: uses 8× data augmentation. Δ Scene Reward: uses scene-change reward (otherwise uses value-based reward). — indicates feature not applicable. *VPG-Equi trains both policies in parallel and is given 2x more training time for a fair comparison.

vention to fully empty the bin. The main policy uses *Pick* actions; the supporting one uses *Pre-Drag* and *Drag*.

Action Discretization: Action spaces are discretized as shown in Table IV. For planar domains, the main policy operates over a 128×128 pixel heightmap with 16 equal rotation bins spanning $[0, \pi]$. For 6-DOF `Drag-Pick`, we additionally discretize height into 36 equal bins over $[0.02\text{m}, 0.20\text{m}]$ and use three rotation axes, each with 16 bins over $[0, \pi]$. The supporting policy's non-grasping primitives (*Push*, *Pre-Push*, *Drag*, *Pre-Drag*) use only planar control (128×128 pixels) without rotation discretization.

A. Setup

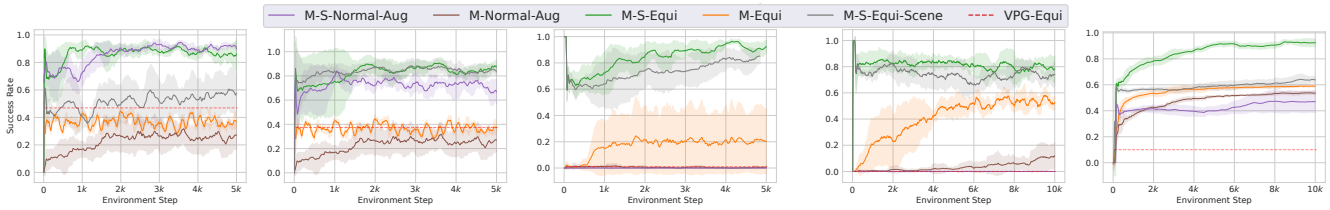
We use two robot embodiments in simulation: a Kuka with a parallel-jaw gripper representing the main policy, and a UR5 with a suction gripper representing the supporting policy (see Fig. 3). Both robots are co-located in the same workspace with inter-robot collision detection disabled. This dual-robot setup enables seamless exploration of primitives with fundamentally different manipulation modes (e.g., parallel-jaw grasping vs. suction-based manipulation) without requiring complex hybrid gripper designs or intricate control mechanisms needed for a single-robot embodiment.

B. Baselines

We compare our proposed **M-S-Equi** agent against several baselines in Table V. Methods using both main and supporting policies are shown in red. **M-Equi** and **M-Norm-Aug** are pre-trained main policies only (5k steps, sufficient for convergence for all tasks), serving as initialization for **M-S-Equi** and **M-S-Norm-Aug**, respectively. M-Norm-Aug replaces equivariant networks with standard CNNs and 8× data augmentation (4 rotations at 90° intervals and 2 flips); M-S-Norm-Aug follows a similar approach to [24]. For ablation, **M-S-Equi-Scene** substitutes our value-based reward with scene-change reward (0.5 if $\sum(s_{t+1} - s_t)$ exceeds a threshold). **VPG-Equi** is an equivariant adaptation of VPG [13] that trains both policies in parallel using scene-change rewards and selects actions by comparing Q-values across primitives (e.g., push vs. pick), unlike our simpler threshold-based selection. Sequential training methods (M-S-*) first pre-train the main policy, then train the supporting policy using the frozen main policy's value function. Since VPG-Equi trains in parallel while our methods train sequentially, we allocate it with twice the training samples for fair comparison. We did not explore parallel training for M-S-Equi, as preliminary experiments showed poor performance—likely because our value-based reward requires a well-trained main policy value function to provide meaningful learning signals.

C. Results

We evaluate the declutter rate ($\frac{\#\text{grasped objects}}{\#\text{total objects}}$) of all agents in Fig. 5, averaged over five random seeds. Since VPG-Equi trains both policies in parallel with twice the training samples, we report the performance of its final checkpoint as a single point. Several key observations emerge from the results. First, two-policy agents (M-S-Norm-Aug and M-S-Equi) achieve 1.3-2× better performance than their single-policy counterparts (M-Norm-Aug and M-Equi), demonstrating the effectiveness of policy collaboration. Note that two-policy methods start at high performance due to the pre-trained main policy (5k steps). Second, comparing



(a) `Block-Pick` (b) `Block-Push` (c) `Drawer-Pick` (d) `Targeted-Pick` (e) `Drag-Pick`

Fig. 5: Declutter rate ($\frac{\#\text{grasped objects}}{\#\text{total objects}}$) averaged over five seeds. Shaded areas show standard deviation. `Block-Pick` and `Block-Push` share the same main policy. VPG-Equi (horizontal line) trains both policies in parallel with $2\times$ samples; only final performance shown.

symmetry-aware and augmentation-based approaches reveals that M-S-Norm-Aug matches M-S-Equi only in simple domains (`Block-Pick` and `Block-Push`), while M-S-Equi significantly outperforms in complex tasks, highlighting the superiority of encoding geometric priors over data augmentation. Third, joint performance correlates strongly with main policy quality, which is expected since the main policy executes most actions. Our method (M-S-Equi) consistently outperforms both VPG-Equi [13] and M-S-Norm-Aug [24] across all domains. Finally, the ablation study shows that scene-change reward (M-S-Equi-Scene) yields inferior performance compared to our value-based reward function, validating our reward design.

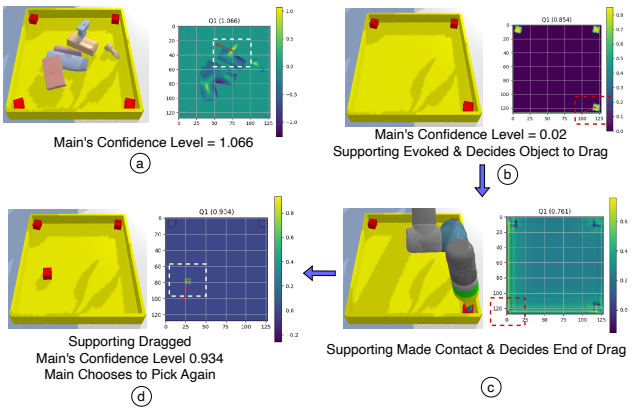


Fig. 6: The main and supporting collaborate in `Drag-Pick`. The supporting policy acts to drag a block at the bottom right corner out so that the main policy can pick successfully.

We illustrate the collaboration between the main and supporting policies in `Drag-Pick` in Fig. 6. For objects away from the four corners, the main policy exhibits high confidence (up to 1.066 in Fig. 6a) to execute successful picks. However, when only corner-positioned objects remain (Fig. 6b), the confidence drops dramatically to 0.02—a result of repeated failed pick attempts during training that taught the policy to have low confidence. At this point, the supporting policy is invoked and selects the block at the bottom-right corner for dragging, as indicated by the peak Q-value in its Q-map (Fig. 6b). After establishing contact with the block, the policy determines the drag endpoint. Interestingly, the Q-map in Fig. 6c reveals that the chosen endpoint is the bottom-left corner rather than an open area as one might expect. This counterintuitive strategy emerges because empty regions lack salient visual features for the agent to attend to. Instead, the policy targets the feature-rich corner, exploiting the physics of the drag motion: the block bounces off the container wall and settles in an open area. This emergent behavior proves effective—after the drag, the block relocates to a position where the main policy can grasp it with high confidence (0.934 at the peak Q-value in Fig. 6d).

VI. LEARNING DIRECTLY ON HARDWARE

For hardware experiments, we perform learning on hardware for `Block-Pick`, `Block-Push`, and `Targeted-Pick`, mimicking the same tasks in simulation.

A. Workspace Setup

Hardware Setup. As shown in Fig. 7, our system comprises a UR5e robot with a Robotiq 2F-140 gripper and a top-mounted Zivid 2+ M130 camera that captures RGB-D images and point clouds. To generate depth observations, we transform the point cloud to world coordinates, filter points outside a cubic working space of $0.4\text{m}\times 0.4\text{m}\times 0.2\text{m}$, and orthogonally project the filtered points onto the table surface to create a 128×128 depth image. Gripper state (open/closed) is queried via the RTDE driver for UR robots. For `Block-Push` and `Targeted-Pick`, after executing a *Pre-Push* action, we superimpose the gripper fingers onto the depth image using the end-effector pose and gripper opening ratio obtained from the RTDE driver.

Object Configurations. `Block-Pick` and `Block-Push` use a single block in the workspace. For `Targeted-Pick`, we maintain separate training and testing object sets, each containing 12 objects (see Fig. 9). Both sets include the same green plastic block (similar to the red block in Fig. 4d) as the target object that must be grasped for task completion.

B. Autonomous Reset and Success Determination

Autonomous Reset Mechanism. On-robot learning requires autonomous episode resets to minimize human intervention for continuous learning. In `Block-Pick` and `Block-Push`, we compute the block's centroid from the depth image, autonomously grasp it, and place it at a random location to initialize the next episode. In `Targeted-Pick`, any held object is placed randomly. However, training

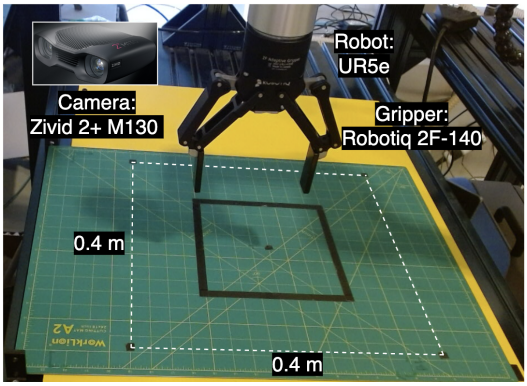
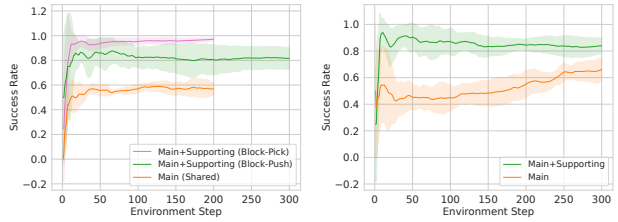


Fig. 7: Hardware setup for robot experiments.

the supporting policy in `Targeted-Pick` requires occasional manual resets to create challenging configurations where the target object is buried or surrounded by distractors—scenarios where the main policy's low confidence necessitates supporting actions.

Success Determination. Task completion is detected through gripper status queries in `Block-Pick` and `Block-Push`. For `Targeted-Pick`, we verify that the correct target (green block) is grasped using a color-based filter applied to RGB images captured from a side-mounted camera that constantly look at the gripper at the home position.

C. Results



(a) `Blk-Pick, Blk-Push` (b) `Targeted-Pick`

Fig. 8: Hardware results using four random seeds. Shaded areas are standard deviations around the mean. The main policy is shared between `Block-Pick` and `Block-Push`.

As shown in Fig. 8, the supporting policy converges rapidly across all domains, requiring only 200-300 environment steps (45 minutes to 1.3 hours of on-robot training), demonstrating the sample efficiency enabled by our symmetry-aware approach. Both `Block-Pick` and `Block-Push` share the same pre-trained main policy, which achieves only 50% success when blocks are positioned inside the inner black square region shown in Fig. 7, motivating the need for supporting actions.

Block-Pick. The joint performance reaches nearly 100%, demonstrating effective policy coordination. When the block falls inside the black square region, the supporting policy picks it up and relocates it outside this region where the main policy can grasp it reliably. Through training, the supporting policy discovers preferred placement locations—typically at workspace corners—where the main policy exhibits high confidence, and consistently commits to these locations.

Fig. 9: Training/testing objects in `Targeted-Pick`.

Block-Push. The supporting policy exhibits different emergent behaviors between simulation and hardware. In simulation, the policy learns a conventional side-pushing strategy, directly pushing the block from its edges. On hardware, however, the policy adapts to a more cautious contact-based approach: resting one or both fingers on the block before executing a push. This strategy ensures reliable contact but occasionally triggers robot safety mechanisms due to collision detection, limiting joint performance to 80%. This issue is absent in simulation, where the same policy can achieve higher without safety-related interruptions.

Targeted-Pick. The joint success rate improves to 80% compared to the main policy's 60%, evaluated on the object sets shown in Fig. 9. The supporting policy exhibits diverse emergent behaviors: it performs long pushes to separate tightly clustered objects, and more frequently executes gentle pushes on the target block or nearby objects to improve target visibility and orientation. The primary failure mode occurs when pushes are sometimes too forceful, inadvertently ejecting the targeted object from the workspace. This issue is minimized in simulation, where objects are contained within bin-like enclosures that naturally prevent most ejections. However, applying this approach in real robot experiments would require additional collision checks, which we opted to avoid in our current implementation.

VII. CONCLUSIONS AND FUTURE WORK

We present a sample-efficient framework for learning multi-step manipulation in challenging industrial grasping scenarios. Our dual-policy architecture demonstrates that complex tasks involving occluded and corner-trapped objects can be effectively automated without manual reward engineering or task-specific heuristics. The integration of C_4 equivariant networks enables rapid on-robot learning within 1-2 hours, representing a practical step toward deployable smart manufacturing systems. The framework's modular design also naturally supports extensibility: new primitives or additional supporting policies can be incorporated without retraining the main policy.

Despite these strengths, several limitations suggest directions for future work. Our method requires a well-trained main policy to provide reliable value-based rewards for the supporting policy. Another limitation is that while symmetry incorporation significantly improves efficiency, real-world tasks may exhibit only partial symmetries. However, encouragingly, recent work [33], [43], [44] demonstrates that equivariant models still outperform non-equivariant alternatives.

REFERENCES

[1] B. Hu, X. Zhu, D. Wang, Z. Dong, H. Huang, C. Wang, R. Walters, and R. Platt, "Orbitgrasp: Se (3)-equivariant grasp learning," in *8th Annual Conference on Robot Learning*, 2024.

[2] H. Huang, D. Wang, X. Zhu, R. Walters, and R. Platt, "Edge grasp network: A graph-based se (3)-invariant approach to grasp detection," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3882–3888.

[3] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 13 438–13 444.

[4] S. Jauhri, I. Lunawat, and G. Chalvatzaki, "Learning any-view 6dof robotic grasping in cluttered scenes via neural surface rendering," *arXiv preprint arXiv:2306.07392*, 2023.

[5] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, no. 26, p. eaau4984, 2019.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] D. Wang, M. Jia, X. Zhu, R. Walters, and R. Platt, "On-robot learning with equivariant models," in *6th Annual Conference on Robot Learning*, 2022.

[8] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 4627–4632.

[9] L. Chang, J. R. Smith, and D. Fox, "Interactive singulation of objects from a pile," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3875–3882.

[10] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, pp. 217–236, 2012.

[11] T. Hermans, J. M. Rehg, and A. Bobick, "Guided pushing for object singulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4783–4790.

[12] M. Danielczuk, J. Mahler, C. Correa, and K. Goldberg, "Linear push policies to increase grasp access for robot bin picking," in *2018 IEEE 14th international conference on automation science and engineering (CASE)*. IEEE, 2018, pp. 1249–1256.

[13] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[14] B. Huang, S. D. Han, J. Yu, and A. Boularias, "Visual foresight trees for object retrieval from clutter with nonprehensile rearrangement," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 231–238, 2021.

[15] M. Kiatos and S. Malassiotis, "Robust object grasping in clutter via singulation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1596–1600.

[16] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, "Deep reinforcement learning for robotic pushing and picking in cluttered environment," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee, 2019, pp. 619–626.

[17] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Multi-step retrieval of a target object occluded by clutter," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1614–1621.

[18] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics Research: The 18th International Symposium ISRR*, 2020, pp. 405–419.

[19] M. Kiatos, I. Sarantopoulos, L. Koutras, S. Malassiotis, and Z. Doulgeri, "Learning push-grasping in dense clutter," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8783–8790, 2022.

[20] M. R. Dogar and S. S. Srinivasa, "A framework for push-grasping in clutter." in *Robotics: Science and systems*, vol. 2, 2011.

[21] M. R. Dogar, K. Hsiao, M. T. Ciocarlie, and S. S. Srinivasa, "Physics-based grasp planning through clutter." in *Robotics: Science and systems*, vol. 8, 2012, pp. 57–64.

[22] I. Clavera, D. Held, and P. Abbeel, "Policy transfer via modularity and reward guiding," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1537–1544.

[23] A. Boularias, J. Bagnell, and A. Stentz, "Learning to manipulate unknown objects in clutter by reinforcement," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[24] P. Ni, W. Zhang, H. Zhang, and Q. Cao, "Learning efficient push and grasp policy in a totebox from simulation," *Advanced Robotics*, vol. 34, no. 13, pp. 873–887, 2020.

[25] Y. Yang, H. Liang, and C. Choi, "A deep learning approach to grasping the invisible," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2232–2239, 2020.

[26] M. Weiler and G. Cesa, "General E(2)-equivariant steerable cnns," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[28] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*. PMLR, 2016, pp. 2990–2999.

[29] D. Wang, C. Kohler, and R. Platt, "Policy learning in se (3) action spaces," in *Conference on Robot Learning*. PMLR, 2021, pp. 1481–1497.

[30] D. Wang, R. Walters, X. Zhu, and R. Platt, "Equivariant q learning in spatial action spaces," in *Conference on Robot Learning*. PMLR, 2022, pp. 1713–1723.

[31] D. Wang, M. Jia, X. Zhu, R. Walters, and R. Platt, "On-robot learning with equivariant models," in *Conference on Robot Learning*. PMLR, 2023, pp. 1345–1354.

[32] D. Wang, R. Walters, and R. Platt, "So (2)-equivariant reinforcement learning," in *International Conference on Learning Representations*, 2022.

[33] D. Wang, J. Y. Park, N. Sortur, L. L. Wong, R. Walters, and R. Platt, "The surprising effectiveness of equivariant models in domains with latent symmetry," in *International Conference on Learning Representations*. International Conference on Learning Representations, 2023.

[34] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, "Spherical cnns," *arXiv preprint arXiv:1801.10130*, 2018.

[35] X. Zhu, D. Wang, O. Biza, G. Su, R. Walters, and R. Platt, "Sample efficient grasp learning using equivariant models," in *Robotics: Science and Systems*, 2022.

[36] H. Nguyen, T. Kozuno, C. C. Beltran-Hernandez, and M. Hamaya, "Symmetry-aware reinforcement learning for robotic assembly under partial observability with a soft wrist," *arXiv preprint arXiv:2402.18002*, 2024.

[37] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[38] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on robot learning*. PMLR, 2018, pp. 651–673.

[39] M. Danielczuk, A. Balakrishna, D. S. Brown, S. Devgon, and K. Goldberg, "Exploratory grasping: Asymptotically optimal algorithms for grasping challenging polyhedral objects," *arXiv preprint arXiv:2011.05632*, 2020.

[40] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[41] D. Wang, C. Kohler, X. Zhu, M. Jia, and R. Platt, "Bulletarm: An open-source robotic manipulation benchmark and learning framework," in *The International Symposium of Robotics Research*. Springer, 2022, pp. 335–350.

[42] H.-S. Fang, C. Wang, M. Gou, and C. Lu, "Graspnet-1billion: A large-scale benchmark for general object grasping," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 444–11 453.

[43] R. Yang, G. Yang, and X. Wang, "Neural volumetric memory for visual locomotion control," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 1430–1440.

[44] D. Wang, X. Zhu, J. Y. Park, M. Jia, G. Su, R. Platt, and R. Walters, "A general theory of correct, incorrect, and extrinsic equivariance," *Advances in Neural Information Processing Systems*, vol. 36, 2024.