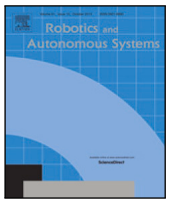
Robotics and Autonomous Systems 197 (2026) 105274

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



Incremental learning from virtual demonstrations and task composition for robotic manipulation☆

Giuseppe Rauso *, Riccardo Caccavale, Alberto Finzi

Department of Electrical Engineering and Information Technology, University of Naples "Federico II", Naples, Italy

ARTICLE INFO

Keywords:
Learning from virtual demonstration
Incremental learning and execution
Structured task composition

ABSTRACT

Developing robotic manipulation capabilities that can be incrementally learned, composed, and transferred across platforms remains a significant challenge. In this work, we propose a modular framework that combines imitation learning from virtual demonstrations with reinforcement learning to incrementally train reusable manipulation primitives, which can be flexibly composed into structured tasks. These policies are learned in simplified virtual environments with a limited number of demonstrations and minimal assumptions about the robotic platform or object properties, promoting generality and cross-platform applicability. The learned primitives are also symbolically represented, enabling their reuse and composition through a Hierarchical Task Network (HTN) planning framework. We validate the framework in scenarios involving structured task generation and execution, combining learned and predefined primitives with realistic manipulators and objects. Experimental results demonstrate high success rates in both primitive and long-horizon tasks as well as effective policy transfer across different simulation environments. By integrating incremental skill acquisition with symbolic task composition, our approach provides a modular and scalable solution for adaptive robotic manipulation.

1. Introduction

Robotic manipulation has made significant progress through reinforcement learning (RL) and imitation learning (IL), enabling agents to acquire complex behaviors from experience or demonstrations [1–3]. However, enabling robots to incrementally acquire and reuse manipulation skills across changing tasks, environments, and platforms remains a key challenge [4–8]. Most learning-based approaches are tailored to fixed task settings [9–11] and often require retraining when task objectives or system configurations change [2,6,12], limiting their applicability in dynamic or long-term autonomous scenarios. To overcome these limitations, we propose a modular framework that combines incremental learning from virtual demonstrations [1,10,12] with symbolic task planning and execution. Our system allows robots to progressively acquire reusable manipulation primitives via a combination of IL and RL, and organize them in a structured repository. These learned skills are then represented symbolically and composed using a Hierarchical Task Network (HTN) planner [13], enabling flexible task generation without end-to-end retraining.

A distinctive feature of the proposed approach is that primitives are trained in simplified virtual environments with a limited number of demonstrations and minimal assumptions about object properties or robot morphology. This promotes generality, cross-platform transfer, and decoupling from robot-specific dependencies. We demonstrate successful deployment of the learned skills in test scenarios that involve more realistic manipulators, diverse object configurations, and a different physics engine, showcasing the system's adaptability. This incremental expansion makes the system suitable for long-term deployment in unstructured or evolving environments, where flexibility and reusability are critical.

In summary, this work presents a novel framework for incremental task learning in robotic manipulation, combining the following key capabilities: (i) learning reusable manipulation skills from virtual demonstrations with minimal assumptions about object properties and robotic platforms; (ii) supporting skill transferability across different robots and simulation environments without retraining; (iii) enabling flexible task composition through symbolic representation and Hierarchical Task Network (HTN) planning and execution.

The remainder of the paper is organized as follows. Section 2 reviews related work on learning from demonstration, incremental skill learning, and task composition. Section 3 presents the overall system architecture. Section 4 details the proposed framework for incremental learning from virtual demonstrations. Section 5 introduces the task

☆ This article is part of a Special issue entitled: 'Planning & Learning' published in Robotics and Autonomous Systems.

* Corresponding author.
E-mail address: giuseppe.rauso@unina.it (G. Rauso).

https://doi.org/10.1016/j.robot.2025.105274

Available online 6 December 2025

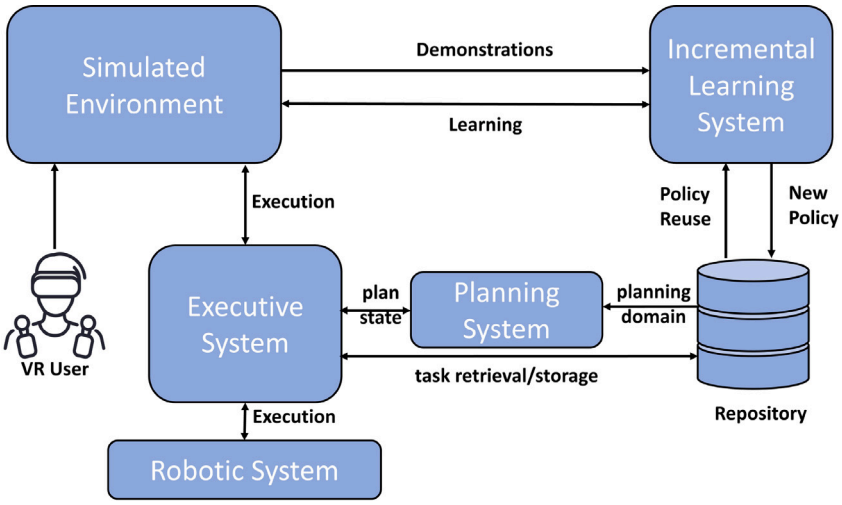


Fig. 1. Overview of the overall incremental learning and execution framework.

composition methodology and discusses structured task generation, execution, and transferability. Finally, Section 6 concludes the paper and outlines directions for future research.

2. Related work

This section reviews related work on learning from demonstration, incremental skill acquisition, and task composition. In particular, we focus on approaches that leverage virtual demonstrations, support incremental skill learning, and enable the composition of long-horizon robotic manipulation tasks.

Different strategies for learning from demonstration in virtual environments have been explored in the literature to guide learning, improve efficiency, and reduce reliance on purely trial-and-error-based approaches [14,15]. Demonstrations can be collected in various ways, including robot teleoperation [16,17], video demonstrations [11,18,19], kinesthetic demonstrations [2,20], or motion capture [1,21]. In this work, we focus on virtual demonstrations collected with a simulated robotic manipulator. A related approach is presented in [1], where the authors propose Demo Augmented Policy Gradient (DAPG), incorporating demonstrations recorded in VR using a motion capture glove, combined with Behavioral Cloning (BC) [3] to support learning. However, their approach is neither incremental nor modular. An alternative, incremental approach to training a robotic manipulator in a virtual environment is proposed in [10], which combines Proximal Policy Optimization (PPO) [22] with Generative Adversarial Imitation Learning (GAIL) [23]. While these approaches successfully leverage demonstrations to guide task learning, they typically do not support task decomposition or the incremental composition of complex tasks.

The compositionality of manipulation tasks has been explored in works like [6], where policies learned with soft Q-learning are combined to generate new composite policies, improving sample efficiency over traditional model-free deep reinforcement learning methods. In this case, multiple policies are combined to create a new policy that jointly solves all constituent tasks. In contrast, we focus on incrementally learning individual primitive policies that can later be composed into structured, long-horizon tasks. The acquisition of complex manipulation tasks is addressed in [24], where a predefined library of primitives is used to learn long-horizon hierarchical policies. Our framework inverts this process: we incrementally learn primitive skills and later compose them using a symbolic planner. This strategy improves modularity, flexibility, and reusability, as new skills can be added and composed without retraining existing ones. More related to the learning from demonstrations paradigm, an integration of imitation learning with task and motion planning (TAMP) to address long-horizon manipulation tasks is proposed in [25]. However, unlike our approach, hierarchical policies are trained to imitate the outputs of a distributed task and motion planning solver. A different strategy for combining TAMP with policy learning is presented in [7], where a task and motion planner is used during training to guide skill acquisition within a reinforcement learning framework. Grounded symbolic operators are associated with policies, which are trained during plan execution using action and state abstractions. In this case, the symbolic plan provides guidance for a Soft Actor-Critic policy learning method without leveraging human demonstrations. Moreover, policies are directly trained and situated within the plan structure. In contrast, our approach first incrementally learns skills from human demonstrations in a generic environment, and then composes them via a planner, keeping the learning and planning stages separate. A complementary approach is proposed in [26], where symbolic programming by demonstration is used to derive action models, automatically extracting preconditions and effects through visual feedback. A related method is presented in [9], where Composable Interaction Primitives (CIPs) are introduced to learn sustained-contact manipulation skills, such as opening a drawer or turning a wheel. However, the learned skills in this case are tightly coupled to the robot's joints, making them specific to a particular robot model. In contrast, our approach abstracts away from the manipulator used during training, supporting skill generalization and reuse across different hardware platforms.

Aligned with our approach, the Mobile-EMBER framework [8] factorizes long-horizon mobile manipulation tasks into primitive visuomotor skills trained with reinforcement learning in simulation, and composes them through symbolic planning. Other work combines task-level planning with reinforcement learning and Bayesian optimization to address contact-rich manipulation tasks [27]. In contrast, our method incrementally acquires reusable skills from VR demonstrations in simplified environments and composes them through HTN planning under minimal assumptions on robots and objects.

3. System overview

In this work, we propose an integrated framework that combines incremental task learning and planning, leveraging human demonstrations in virtual environments, reinforcement learning, and hierarchical task composition. The goal is to support lifelong autonomy by enabling incremental task teaching and training in simulation, along with the flexible generation of structured behaviors composed of both learned and predefined actions or methods.

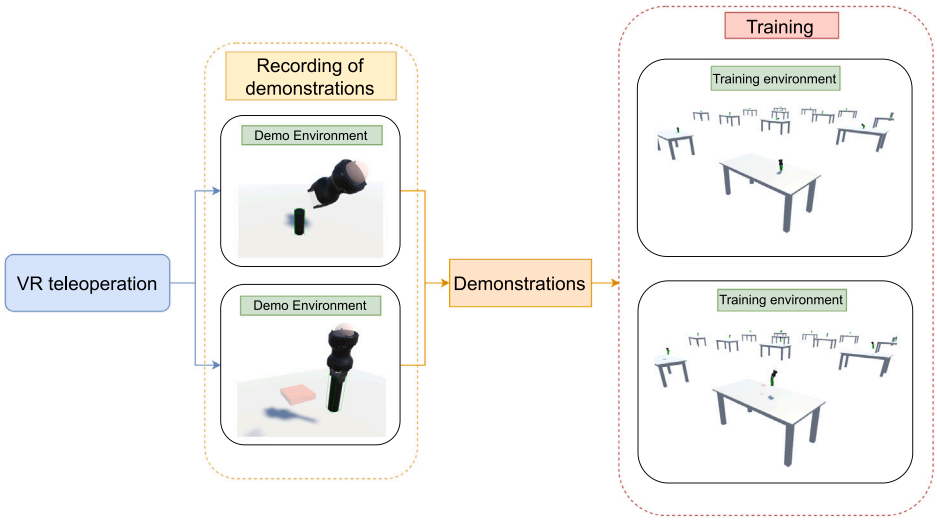


Fig. 2. General pipeline scheme for training the grasp-and-lift and place policies through reinforcement and imitation learning. First, the human operator teleoperates the gripper in the simulated environment to perform the tasks, generating a dataset of demonstrations. The agents are then trained in the same environments by combining reinforcement and imitation learning, leveraging the collected data.

A distinctive feature of the approach is imitation learning, enabled by demonstrations provided through virtual reality simulation. Additionally, the system is designed to rely on minimal information about the object and the robotic system, ensuring that learned policies can be reused across different configurations and applied to previously unseen objects.

The overall framework is illustrated in Fig. 1. The learning and execution pipeline operates as follows: first, the operator provides task demonstrations in a virtual environment, which are then used to train the corresponding policy within the simulation. The learned policies are stored in a repository along with their specifications, making them available for composition and deployment via task planning and execution. Specifically, stored methods and primitive actions can be leveraged by the planning system to generate structured activities, by the execution system to directly perform the generated tasks, or by the incremental learning system to progressively train more complex behaviors. In the proposed framework, incrementality is achieved in two complementary ways. First, complex behaviors can be trained incrementally by building on previously acquired policies. Second, the skill repository can be expanded incrementally, where newly learned behaviors are combined through the planning system to produce increasingly complex capabilities.

4. Incremental learning from virtual demonstrations

The proposed incremental learning methodology is illustrated considering two manipulation tasks: *grasp-and-lift* and *place*. We describe the training environments, actions, observed state definitions, rewards, and demonstration recording, along with the reinforcement and imitation learning algorithms employed. The action model and teleoperation methodology for controlling the simulated gripper — and thus recording demonstrations — are consistent across both tasks and can be found in Sections 4.3 and 4.4. Then, for each environment, we present the observed state and reward model.

4.1. Simulated scenario

The simulated robotic setup consists of a *Robotiq Hand-E* gripper, which is suspended and detached from a robotic arm, meaning no specific arm model is assumed. This design choice enhances the generalizability of the approach, enabling adaptation to various manipulator platforms. Task demonstration data are collected via teleoperation, where the human operator controls the end-effector movement in a simulated virtual environment. For this purpose, a Meta Quest 2 virtual reality headset is used. The simulation environment is implemented in *Unity 2022.3.47f1*, while training is carried out using the *ML-Agents* [28] framework, which provides native support for the learning algorithms used in this work and facilitates their integration. The complete pipeline, encompassing the recording of demonstrations and the subsequent training within the same environment, is illustrated in Fig. 2.

4.2. Models and methods

For policy training, we build on the approach proposed in [12], combining reinforcement learning and imitation learning using Proximal Policy Optimization (PPO), Generative Adversarial Imitation Learning (GAIL), and Behavioral Cloning (BC) algorithms. PPO operates as an on-policy RL method, ensuring that policy updates are gradual and avoiding large, disruptive changes through a surrogate objective function that clips the ratio between the old and new policies. Demonstrations of manipulation tasks are utilized by both GAIL and BC; in the first case, GAIL generates an intrinsic reward using a discriminator network to distinguish expert trajectories from those of the policy, learning in an adversarial manner. In the second case, BC directly guides the policy to imitate the demonstrations by minimizing a loss between the actions predicted by the policy and those performed by the expert. BC is particularly useful in the early stages of training to quickly align the policy with expert behavior. As a result, the influence of BC is reduced throughout training, allowing PPO updates — driven by the environment reward and GAIL's intrinsic reward — to become more prominent as training progresses. In this setting, GAIL's reward not only supports imitation learning generalization, but also compensates for the sparsity of the environment reward in PPO. In the proposed methodology, for the presented learning scenarios, in addition to an initial phase where all three algorithms are used simultaneously, subsequent refinement phases are included that leverage only the environment reward and the GAIL reward with PPO. Further details are provided in the following sections.

4.3. Simulated actions

As mentioned above, in the proposed simulated scenario, the gripper is floating and, at each physics step, the agent moves the gripper

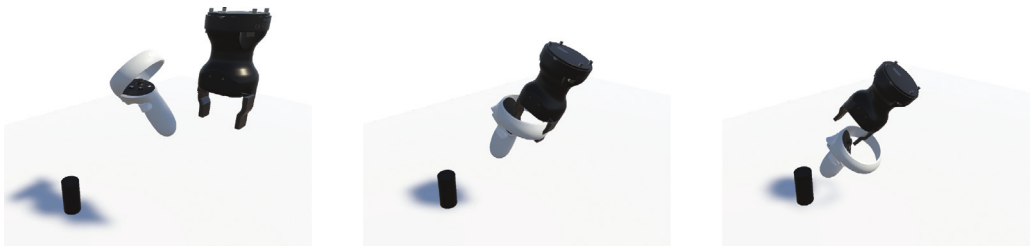


Fig. 3. Example of teleoperation in virtual reality during a demonstration phase using the controller to move the gripper and grasp an object. The corresponding changes in position and rotation are recorded as demonstration data, while the gripper follows these targets through motions induced by the simulated physics.

toward a target position $P_{\text{target}} \in \mathbb{R}^3$ and target rotation $q_{\text{target}} \in \mathbb{R}^4$ within predefined limits. The agent controls the displacement and rotation of this target position by specifying the change relative to the current target values, while the resulting linear and angular velocities that drive the gripper's motion are induced by the simulated physics. Additionally, the agent controls the gripper's state (open or close) using a discrete binary value $C \in \{0, 1\}$, where $C = 1$ indicates closing the gripper and $C = 0$ indicates opening it, at a fixed speed. The gripper's opening/closing action is integrated with the movement commands. More specifically, actions — whether generated by the policy or collected from expert demonstrations — are denoted by the tuple:

$$a = (\Delta P_{\text{target}}, \Delta q_{\text{target}}, C)$$

where $\Delta P_{\text{target}} \in \mathbb{R}^3$ represents the displacement in position, $\Delta q_{\text{target}} \in \mathbb{R}^4$ is the change in orientation (as a quaternion), and $C \in \{0, 1\}$ is the binary signal for gripper action. The magnitude of the displacement ΔP_{target} is constrained by a maximum value $M_{P_{\text{target}}}$, and the maximum allowed change in orientation Δq_{target} is limited to an angle M_θ, where θ is the angle of Δq_{target} in angle-axis representation.

4.4. Virtual demonstration collection

Demonstrations are acquired by an operator using the virtual reality (VR) setup. The operator uses VR controllers to manipulate the gripper's position and orientation, and to activate or deactivate the grasp mechanism (see Fig. 3). Let $P_{\text{controller}} \in \mathbb{R}^3$ and $q_{\text{controller}} \in \mathbb{R}^4$ denote the position and orientation of the VR controller. The target displacement and delta rotation provided as the demonstration action are computed as:

$$\Delta P_{\text{target}} = P_{\text{controller}} - P_{\text{target}}, \quad \Delta q_{\text{target}} = q_{\text{target}}^{-1} q_{\text{controller}}$$

The grasp control signal $C \in \{0, 1\}$ is directly triggered by the operator through button presses on the VR controller, where $C = 1$ indicates a closed gripper (grasp activated), and $C = 0$ indicates an open gripper.

This teleoperation mechanism is integrated into all simulated environments and is agnostic to the specific task. As such, it can be used to generate demonstration trajectories for grasping, lifting, placing, and future extensions without the need for environment-specific adjustments. Demonstration data are stored as sequences of observations and actions, which can be used to initialize policies, pretrain models, or augment the training dataset. While actions are consistent across tasks, observations vary depending on the environment and are described in the following sections. The demonstration episodes were recorded with randomized initializations of the gripper and object positions on the table to ensure sufficient variability. Further details on these randomizations are provided in Sections 4.5 and 4.6.

4.5. Grasp-and-lift policy training setup

This section describes the training setup for the grasp-and-lift policy, assuming the agent is equipped with a generic gripper and has minimal prior knowledge about the objects to be manipulated. To promote generalization, we train the agent using cylindrical objects. The agent only receives the 3D bounding box of each object, without any explicit information about its category. By varying the dimensions of these cylinders, we can implicitly represent a wide range of everyday items: slender and elongated cylinders may resemble handles or tubes, while thicker and taller ones can approximate bottles. Smaller cylinders allow the agent to generalize its grasping skills to compact objects such as cubes or other similarly shaped items. The training can also be specialized for specific shapes or extended to other object types when required.

The training environment is illustrated in Fig. 4 (left). In this setup, we train the agent to execute different grasp-and-lift strategies. Specifically, as in [12], we focus on two types of grasps: a top grasp (Fig. 4, center) and a side grasp (Fig. 4, right). Each grasp type requires the agent to reach and align with specific regions of the object to perform a successful grasp.

Training proceeds as follows. At the beginning of each episode, the gripper is oriented downwards and randomly positioned within a predefined 3D region above the table. An object is placed at a random location within a defined area on the table, with its radius, height, and mass sampled from predefined intervals. The agent is also informed of the required grasp type (either top or side). The task is considered successful if the agent grasps the object and lifts it to a target height. In our setup, the gripper region spanned $0.3\,\text{m} \times 0.06\,\text{m} \times 0.23\,\text{m}$, while the object placement area covered $0.6\,\text{m} \times 0.297\,\text{m}$ on the table surface.

In the following, we detail the learning setup, including the agent's observations, action space, reward structure, and the use of demonstrations.

4.5.1. Observations

To enable the agent to grasp and lift an object, it receives observations that provide spatial and dynamic information about both the object and the end-effector. In line with our goal of generality and transferability, we limit the object information to its 3D bounding box, which encodes its approximate shape and position. Specifically, the observed spatial features are the relative positions between the eight corners of the object's bounding box and the end-effector position, defined as the midpoint between the gripper's fingers. These features allow the agent to infer alignment and proximity between the gripper and the object.

To promote applicability across different grippers and robotic hands, the agent also observes a binary grasp signal indicating whether the object is currently grasped. In our setup, a grasp is detected when the gripper is closing, contact with the object is identified, and the contact force on the finger joints exceeds a predefined threshold. However, the specific detection rule can vary with the gripper hardware, while the policy only observes the binary grasp indicator, thus ensuring consistency and reusability across setups.

Fig. 4. Example of an initial configuration in the environment for training the grasp-and-lift task, with the gripper positioned above the cylinder and oriented downward (left), execution of a top grasp (center), and execution of a side grasp (right).

Finally, the agent observes the type of grasp to be performed — either top or side — encoded as a one-hot vector. This choice avoids introducing a spurious notion of ordinality between grasp types. In addition, the one-hot representation provides separated neural channels for each grasp type, increasing the internal expressiveness of the policy and facilitating task discrimination.

Formally, the observation tuple is:

$$o = (d_1, \dots, d_8, P_{\text{obj}}, q_{\text{obj}}, v_{\text{obj}}, \omega_{\text{obj}}, P_{\text{ee}}, q_{\text{gb}}, v_{\text{gb}}, \omega_{\text{gb}}, G, T)$$

where $d_1, \dots, d_8 \in \mathbb{R}^3$ are the relative bounding box corners, P, q, v, and ω denote positions, orientations (as quaternions), linear, and angular velocities for the object and gripper base, respectively. P_{ee} denotes the position of the end-effector point, located between the gripper fingers. $G \in \{0, 1\}$ indicates whether the object is grasped, and $T \in \{0, 1\}^2$ encodes the grasp type.

4.5.2. Reward

The reward function is designed to assign positive feedback to successful grasping and lifting behaviors. Specifically, a positive reward R_{grasp} is given when the agent correctly grasps the object, based on the criterion defined for a successful grasp (see Section 4.5.1) and thus indicated by the binary signal $G = 1$ observed by the agent. Upon successfully lifting the object above a specified height threshold h, and while maintaining a stable grasp, the agent receives an additional reward R_{lift} assigned for task completion. No reward is provided in case of failure, which may occur due to premature grasp release, collisions with the environment, activation outside designated regions, or exceeding the maximum number of allowed steps. The lifting reward is granted only if the object remains grasped after surpassing the height threshold. An episode terminates either upon successful task completion (i.e., lift with stable grasp) or upon any failure event.

4.6. Place policy training setup

In this section, we describe the methodology used to train the place policy, which enables the agent to position a grasped object into a specified target zone.

At the beginning of each episode, the object is assumed to be already securely held by the gripper, which remains closed throughout the task. To ensure realistic initial states, a dataset is constructed from successful executions of the grasp-and-lift policy. This dataset includes gripper positions and orientations, as well as object properties (size and orientation) at the moment of lift. For each training episode, a sample from this dataset is used to initialize the grasped object configuration along with the gripper orientation, while the gripper's position is randomly varied within the same 3D region used for the grasp-and-lift task.

The placement target is defined as a region in space above the table surface, which does not necessarily lie directly on the table (Fig. 5, center). In our setup, it is randomly positioned above the table at a distance of 0.2 m from the initial location of the grasped object. The goal is to vertically place the object so that a specific portion of its base — highlighted in red (Fig. 5, left) — makes contact with the designated target zone (Fig. 5, right). To succeed, the agent must maintain this contact continuously for a fixed number of steps, denoted as n_{place}. Since the object must always be placed in the same orientation, differentiating between grasp types is unnecessary in this scenario. Instead, the focus is on precise alignment and positioning within the target region.

As in previous tasks, in the following we detail the agent's observations, reward formulation, and demonstration generation in this environment. As for the action representation, it remains unchanged and follows the formulation described in Section 4.3.

4.6.1. Observations

To support learning in the placement task, the agent receives a structured observation designed to capture spatial relationships relevant to the task, while maintaining a limited and generalizable representation of the object.

Unlike the grasping phase, in this case we assume the object is already held by the gripper at the beginning of each episode. Since the object has already been taken, the agent no longer requires relative position vectors between the gripper and the object, instead directly observes the absolute positions of the eight vertices of the object's bounding box. This abstraction allows for consistent geometric encoding regardless of object type or pose.

On the other hand, to control the placement policy, the agent also observes the absolute position of the designated target zone. Additionally, a compact scalar feature is introduced to provide an explicit measure of proximity. Specifically, this is obtained as the average distance between the base of the object (i.e., the lowest four bounding box points) and the center of the target zone. This promotes alignment during fine-grained control near the placement site, while preserving generality across object shapes and target geometries. The remaining features align with those used in the grasp-and-lift task (Section 4.5.1), including object orientation and dynamics (linear and angular velocity), gripper pose and motion, and a binary grasp signal indicating contact stability. More formally, the agent's observation vector is defined as:

$$o = (P_1, \dots, P_8, d_{\text{mean}}, q_{\text{obj}}, v_{\text{obj}}, \omega_{\text{obj}}, P_{\text{place}}, P_{\text{ee}}, q_{\text{gb}}, v_{\text{gb}}, \omega_{\text{gb}}, G),$$

where $P_i \in \mathbb{R}^3$ are the bounding box points, d_{mean} is the average distance from the bounding box base to the target, and the remaining terms are as described previously.

4.6.2. Reward

The reward formulation for the placement task is similar to the one used in the grasp-and-lift phase, with adaptations introduced to enable multi-step task evaluation. In this scenario, the agent is rewarded for maintaining the object in a stable and well-aligned position within a designated target zone.

At each time step, a placement reward R_{place} is assigned if two conditions are met: (i) the base of the object is in contact with the target region (see Fig. 5), and (ii) the object's orientation aligns within

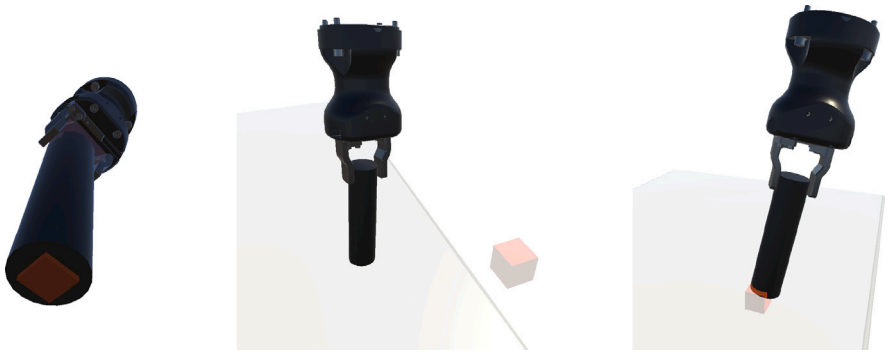


Fig. 5. Cylinder area (left, in red) to be placed in the target zone (center and right, in red) to complete the place task.

a predefined angular threshold with the upright direction of the target zone. This ensures both positional and angular accuracy during placement. The task is considered successfully completed if these placement conditions are satisfied for a consecutive number of steps, denoted n_{place}. When this condition occurs, the agent receives a cumulative reward proportional to the number of successful steps.

Failure conditions include those of the grasp-and-lift task, with the addition of a grasp stability constraint: the agent must maintain a closed grasp throughout the placement phase. Releasing the object prematurely, losing alignment, or exceeding the maximum episode length all result in early termination, with the total reward reflecting only the progress made up to that point. This encourages the agent to not only reach the target zone but also to maintain stable and precise object placement over time.

4.7. Training and evaluation

To train the grasp-and-lift and place policies, 50 demonstration episodes were recorded for each task in the described environments. These demonstrations were collected using the teleoperation methodology for the gripper outlined in Section 4.4. For grasp-and-lift, 25 episodes corresponded to top grasps and 25 to side grasps. For place, instead, the 50 episodes were generated by initializing the gripper with grasps sampled from the dataset of successful executions of the grasp-and-lift policy (see Section 4.6). As specified in Section 4.2, a combination of reinforcement learning and imitation learning was employed. Specifically, we used the PPO algorithm in combination with GAIL and BC.

Fig. 6 illustrates the overall training pipeline for both tasks, which were learned sequentially. Each task was first subjected to an initial training phase combining PPO, GAIL, and BC, followed by one or more policy refinement phases using PPO and GAIL. The evolution of environment reward across these incremental training stages for both policies is shown in Fig. 7.

During initial training (Training 1 in Fig. 7), the environment reward (as described in Sections 4.5.2 and 4.6.2) was combined with the GAIL reward, weighted at 0.025. In addition, BC was applied with a strength of 0.125. The BC strength was gradually reduced to zero by the end of training. This progressive reduction diminished BC's influence on policy updates, allowing PPO and GAIL to drive learning in later stages. For instance, in Fig. 7, Training 1 (top-left) is mainly driven by BC and the GAIL intrinsic reward, before the policy starts achieving positive environmental rewards in later stages. Each initial training phase lasted 20 million steps, with a starting learning rate of 2.5×10^{-4}, linearly decayed over time.

After initial training, each task underwent refinement phases (Training 2 in Fig. 7) using PPO and GAIL, the latter with the same reward weight. Each refinement phase consisted of 20 million steps and used a lower initial learning rate of 5×10^{-5}.

For the grasp-and-lift task, training was organized in two stages (see Fig. 6, top). The first stage used cylinders of fixed height and included three consecutive phases, with the third one (Training 3 in Fig. 7, top-left plot) run at a further reduced learning rate of 1×10^{-5}. The second stage introduced randomized cylinder heights and consisted of two additional refinement phases under these new conditions.

In contrast, the place task was trained directly with randomized cylinder heights, without fixed-height pretraining (see Fig. 6, bottom). Two training phases were conducted under these conditions, each lasting 20 million steps and using the same hyperparameters as in the grasp-and-lift task, with the reward formulation defined in Section 4.6.2.

The algorithms were implemented using the ML-Agents framework. Batch size (512), buffer size (64,000), and time horizon (512) were fixed across all training sessions. Threading and observation normalization were enabled. The neural network architecture consisted of three fully connected hidden layers, each with 512 neurons. Hyperparameters were selected through an initial exploratory stage of automated search, limiting the training length, followed by manual tuning based on observed learning performance. All other hyperparameters were kept at their default values.

The training was performed on a machine equipped with an Intel Core i9-10980XE processor, 64 GB of DDR4 RAM, and an NVIDIA Quadro RTX 4000 GPU with 8 GB of VRAM. The cumulative training time for the grasp-and-lift task is approximately 43 h (3 training sessions with fixed cylinder height and 2 with random height), while for the place task, it is around 12 h (2 training sessions), using 64 parallel environments to fill the transition buffer efficiently.

At the end of the training, success rates were calculated for both tasks. These were collected in the same environments described previously, across 200 episodes (randomly selecting at the beginning whether to grasp from above or from the side) and averaged over 5 different seeds. For the Grasp-and-lift task, we observed $99.6\% \pm 0.45$ for top grasping and $99.2\% \pm 0.73$ for side grasping with variable cylinder height. For the place task, the success rate was $98.1\% \pm 0.32$. The reported values are confidence intervals of 95%.

5. Task composition and transferability

In this section, we explore how the learned tasks can be flexibly combined and executed to accomplish structured, goal-oriented activities. We also show how policies trained in a virtual simulated environment can be transferred and executed in a more realistic simulation environment, and on different robotic platforms.

5.1. Testing scenario: Policy transfer and task compositionality

In order to test the portability of the trained policies and their generalization to different manipulators, we developed a CoppeliaSim environment similar to the one used in Unity. However, in this environment, the gripper (different from the one used in Unity) is mounted on a robotic arm equipped with inverse kinematics for joint rotation.

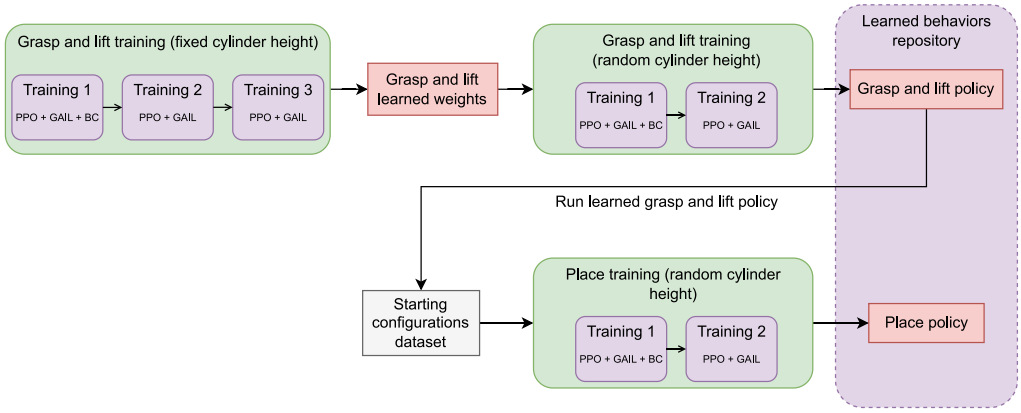


Fig. 6. Training scheme for the grasp-and-lift and place policies. First, the agent is trained for the grasp-and-lift task with a fixed cylinder height. Then, starting from the obtained policy and value function weights, a new training phase is performed by setting the cylinder height to a random value within a given range at the beginning of each episode. Finally, to obtain the policy for the place task, a dataset of final configurations from the grasp-and-lift task is generated using the previously learned policy. These configurations serve as the initial states for training the place task.

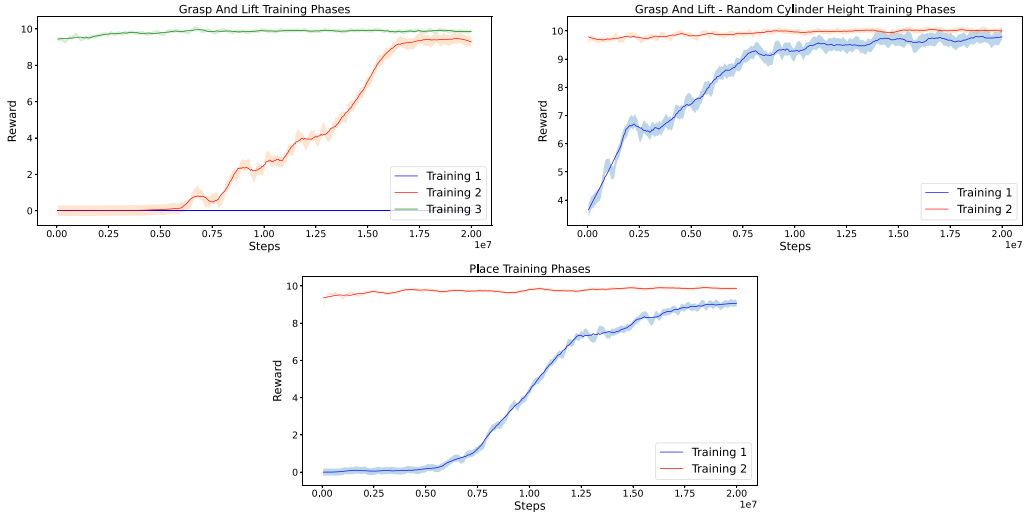


Fig. 7. Evolution of the reward during different training phases for both the grasp-and-lift and place policies: grasp-and-lift task with a fixed cylinder height (top-left image); grasp-and-lift task with random cylinder height (top-right image); place task (bottom image).

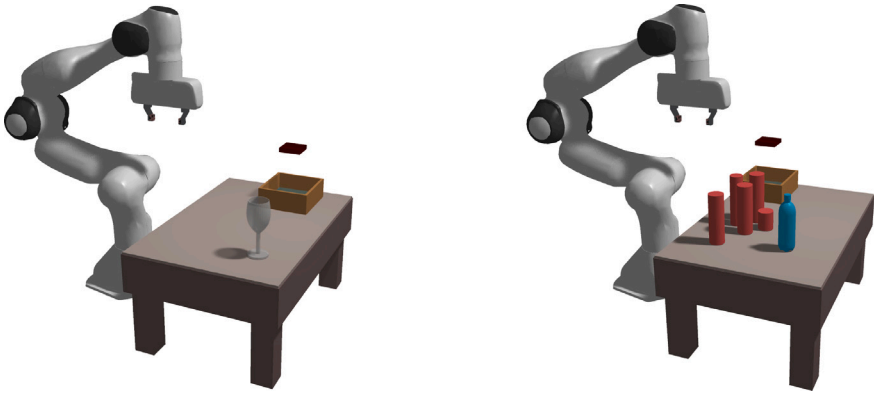


Fig. 8. Environments developed in CoppeliaSim for pick-and-place (Left) and for the clear-and-place task (Right), with randomly positioned objects and box, using a Franka Emika Panda robotic arm and Franka gripper.

The policies trained in Unity were integrated into the CoppeliaSim simulation and executed to perform the tasks for which they were trained. In the new simulated environment, we discuss task generation and execution by considering two increasingly structured tasks. The first is a typical *pick-and-place* task, where the two trained policies, for grasp and lift, and for place, are combined with already available motion primitives. This scenario is shown in Fig. 8 (left), while an example of task execution is shown in Fig. 9.

The second task, referred to as *clear-and-place*, involves multiple cylinders placed on the table as obstacles, along with a bottle, the target object to be grasped. The robot's objective is to first remove all cylinders by placing them outside the table's boundaries, then grasp the bottle and place it inside a box. This task requires a sequence of pick-and-place actions, which include moving the cylinders off the table, performing intermediate repositioning, and finally placing the bottle in the box. The environment developed for this task is shown in Fig. 8 (right), and an example of a cylinder removal execution is shown in Fig. 10.

5.2. Planning domain specification

To enable activity composition and monitoring, the system's procedural memory (i.e., the repository) must explicitly represent both predefined and learned operators and methods. We adopt a Hierarchical Task Network (HTN) framework, where complex tasks are recursively decomposed into sub-tasks until reaching primitive operators [13]. In this framework, a *planning problem* is defined by a set of tasks $t \in T$, which can be either primitive or compound, that must be accomplished from an initial state. The system is provided with a representation of a *planning domain*, which defines a set of primitive operators $o \in Op$ for executing primitive tasks and a set of methods $m \in Me$ that specify how to achieve compound tasks.

Each operator o is represented in a STRIPS-like formalism, defined by its preconditions and effects. Each method is a triple $m = (t, \delta, d)$, where t is a task name, δ is a precondition for the method, and d is a task network describing a possible way to achieve t through a structured combination of sub-tasks. The world state is represented symbolically in a language $\mathcal{L}$, which consists of a set C of constants and V of variables representing objects, and a set P of predicates capturing relationships among these objects. A given state of the world is characterized by a list of instantiated predicates $p(c_1, \ldots, c_n)$, with $c_i \in C$, which describe the properties that hold in that state. Each operator has preconditions prec(o) specifying required predicates that must hold for o to be executed, and effects eff(o) that define how the world state changes after its execution. Effects are given as a pair (γ^+, γ^-), where γ^+ (add-list) includes predicates that become true, and γ^- (delete-list) includes predicates that no longer hold.

In this context, a newly trained policy must be integrated into the planning domain as a new operator symbol o', with appropriately defined preconditions prec(o') and effects eff(o'). Once new primitive operators are introduced, additional compound tasks and methods can also be defined to suitably integrate and exploit the new skills into the system.

The hierarchical structure enables the definition of increasingly complex tasks. By leveraging this planning domain, an HTN planner can generate feasible action sequences, ensuring that both predefined and learned operators are effectively integrated into the planning and execution process.

5.2.1. Manipulation scenario

In the proposed scenario, a robotic manipulator is tasked with picking up and placing objects on a table. The planning domain specification represents the trained policies *grasp-and-lift* and *place* as primitive operators `pickUp(mode, obj)` and `place(obj, loc1, loc2)`. Here, `obj` denotes the target object, `mode` specifies the grasp type (e.g., top or side), while `loc1` and `loc2` represent initial and placement locations. Newly trained operators incrementally extend a predefined repository of primitive operators. To exemplify this scenario, in our setting, we assume the robot is already endowed with the operator `move(loc1, loc2)`, which moves the end effector between locations, as well as the operator `release(obj)`, which opens the gripper to release a grasped object.

The planning domain is defined using predicates to describe the robot's state, the state of the table, and the objects to be manipulated. The table is represented as a discretized grid of locations, which can either be `free(loc)` or occupied by an object, denoted as `at(obj, loc)`. Objects may be `movable(obj)` or static (e.g., a container like a box, or objects for which suitable manipulation policies are not available). The robot's end-effector can move across locations, and its current position is denoted by `at(robot, loc)`. Additionally, the predicate `holding(obj)` specifies whether the robot is holding an object. Locations can be `reachable(loc1, loc2)` if they are accessible for the robot, or obstructed by obstacles otherwise. We also introduce predicates to denote whether objects are accessible from the top, `front_free(obj)`, or from the side, `left_free(obj)` or `right_free(obj)`.

In this formalism, each operator is defined by its preconditions and effects. For the trained operators, the action `pickUp(mode, obj)` requires the following preconditions: the object must not already be held (`not holding(obj)`), it must be movable (`movable(obj)`), and it must be reachable from the robot's current position (`at(robot, loc1)`, `at(obj, loc2)`, and `reachable(loc1, loc2)`). Similarly, the action `place(obj, loc1, loc2)` requires that the robot is already holding the object (`holding(obj)`), that the destination is reachable from the robot's position (`reachable(loc1, loc2)`), and that the robot itself is at `loc1` (`at(robot, loc1)`).

Furthermore, trained primitives op are associated with operator-specific preconditions $prec_{op}$ which define additional constraints that align with the conditions present in the training setup. For instance, both `pickUp` and `place` also require that the target location is close, i.e., `near(loc1, loc2)`, to ensure the robot is in a comfortable state for policy deployment.

The effects for these primitives are specified as follows. For `pickUp(mode, obj)`, the positive effects (γ^+) include `holding(obj)` and `free(obj_loc)`, while the negative effect (γ^-) removes `at(obj, obj_loc)`. Similarly, for `place(obj, loc1, loc2)`, the negative effect (γ^-) removes `at(robot, loc1)`, while the positive effect (γ^+) is `at(robot, loc2)`. For the non-trained operators, `move(loc1, loc2)` requires the robot to be at the initial location and the two locations to be reachable, i.e., `at(robot, loc1)` and `reachable(loc1, loc2)`, having the effect of removing `at(robot, loc1)` (negative) and adding `at(robot, loc2)` (positive) from the state. The `release(obj)` operator requires `holding(obj)` as a precondition, with its positive effects including `at(obj, loc2)`, while its negative effect removes `holding(obj)`.

Once primitive operators are defined, the planning domain includes methods that define strategies for executing compound tasks involving multiple primitive actions. These methods specify different ways to sequence and combine operations to accomplish a task.

Notice that, in this setting, methods can also be exploited to effectively deploy trained operators. For example, in the pick-and-place scenario, to induce a grasp from a specific side, we introduce methods `m_take_right(obj)` and `m_take_left(obj)`, which move the robot's end-effector to the right or left side of the object before executing `pickUp` in `side` mode. This way, even if the trained *Grasp-and-lift* policy does not explicitly distinguish between left- and right-grasping, methods can be defined to ensure that the robot moves into an appropriate pre-grasp position, enacting the desired behavior. Similarly, a method may directly include preparatory movements to satisfy the preconditions of a learned policy. For example, since the operator `place` requires `near(loc1, loc2)` to hold, a `leave(obj)` compound task can be defined to first execute a `move` action, bringing the robot close enough to the placement location to meet this requirement, followed by a `release` action.

5.3. Pick-and-place task

Given the planning domain introduced above, we now consider a simple pick-and-place task designed to illustrate the system in action within an obstacle-free scenario. The robot's goal is to pick up objects of different types and place them inside a box.

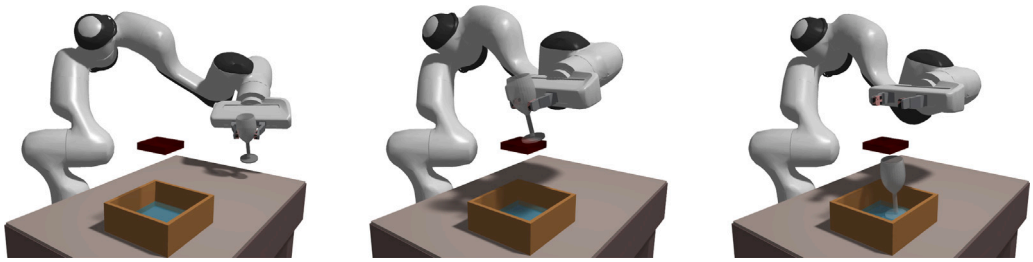


Fig. 9. Execution of the pick-and-place task, where the object is lifted (left), moved to the target position (center), and released into the box (right).

5.3.1. Simulation scenario

In the simulated scenario, both the object and the box are positioned on a table within the robot manipulator's workspace (see Fig. 9). The placement area is divided into two regions, and at the beginning of each episode, both the object and the box are placed at random locations within their respective regions such that if the object is placed in one zone, the box is placed in the other. To evaluate the execution of the generated plans in simulation, we tested object manipulation with a set of shapes different from those used during policy training. Specifically, we considered four types of objects: *cylinder*, *box*, *wine glass*, and *water bottle*. While the first two are simple geometric shapes already available in the CoppeliaSim simulator, the latter two were imported from the open dataset proposed in [29], which is a slightly modified version of ContactDB [30].

5.3.2. Task specification

From a planning perspective, the `pick-and-place` task is decomposed into two main sub-tasks: `take(obj,mode)` and `leave(obj, loc)`. The `take(obj, mode)` task incorporates the learned `pickUp` primitive, which can be executed in different manners depending on the specified `mode`. This mode can be either `top`, for top grasping, or `lat`, for lateral grasping. Lateral grasping is handled by the methods `m_take_left(obj,lat)` and `m_take_right(obj,lat)`, depending on whether the object is grasped from the left or right side. Conversely, the method `m_take_top(obj,top)` is used for top grasping. These methods are defined as follows:

```
Method: m_take_left(obj,mode)
   Pre: (mode=lat,movable(obj),left_free(obj), at(robot,loc1), at(obj,loc2) )
   Sub: (move(loc1, left(loc2)), pickUp(side,obj))

Method: m_take_right(obj,mode)
   Pre: (mode=lat, movable(obj), right_free(obj), at(robot,loc1), at(obj,loc2) )
   Sub: (move(loc1, right(loc2)), pickUp(side,obj))

Method: m_take_top(obj,mode)
   Pre: (mode=top, movable(obj), front_free(obj), at(robot,loc1), at(obj,loc2))
   Sub: (move(loc1, front(loc2)), pickUp(top,obj))
```

As described above, each method begins with a `move` operation that repositions the robot's end-effector near the object from the selected direction. This is followed by a `pickUp` operation, which executes the appropriate grasping strategy based on the specified mode.

On the other hand, the `leave(obj, loc)` task can be executed in a single `m_leave(obj, loc)` method.

```
Method: m_leave(obj, loc)
   Pre: (holding(obj), at(robot,loc1), near(loc2,loc))
   Sub: (move(loc1,loc2), place(obj,loc2,loc), release(obj))
```

This method consists of a `move` operation toward the box location, followed by the sequential execution of the `place` and `release` operators. Thus, a generated plan for `pick-and-place` consists of five main steps. At first, the robot moves toward the object in order to enable the `pickUp` policy, reaching a goal pose that is either beside (on the left or on the right) or in front of the object, depending on the selected grasping mode (left, right, or top). Secondly, a `pickUp` operator is executed, where the robot grasps and lifts the reached object following the trained policy. The grasped object is then carried toward the box by means of a second `move`; the `place` primitive is executed followed by the `release`.

5.3.3. Evaluation

Given the task and domain specifications defined above, we can consider various object arrangements in the initial state, generate plans, and execute them, including two modes of grasping, top and lateral, to assess their execution performance. For each of the two grasping modalities, a total of 200 executions of the pick-and-place task have been performed (50 runs for each object type) over randomly generated tabletop configurations (i.e., 10 times for 5 different seeds). For each seed, the mean success rate is computed over 10 episodes, after which the results are averaged across the 5 seeds. The tables report the corresponding 95% confidence intervals.

The collected results are presented in Table 1 for top grasping and in Table 2 for lateral grasping over the different types of objects. In the tables, we report the success rate of each primitive operator involved in the generated plan, excluding the first `move` toward the target object, which is omitted because always successful. Notice that when a task execution fails on an intermediate stage, also the following ones are considered unsuccessful, thus the last column of both tables (`release`) collects the overall success rates of the experiments. For both modalities (side and top) over the first 3 objects (cylinder, wine glass, and cube) the success rate is always over 82%, while there is a drop in performance when the robot handles the water bottle from the top (64%). This was expected, as top grasping of bottles is known to be particularly challenging because of the odd shape of the bottleneck (similar difficulties have also been reported in [12]). The success rates for the lateral grasp, starting from the `place` action, are generally lower than those for the top grasp. This is because in some cases, the movements generated by the learned policy conflict with the inverse kinematics used for the robotic arm, leading to undesired positions or singularities, which result in jerky and incorrect motions. This issue is particularly evident in the policy for the `place` action, which is trained to perform feasible rotations without considering the robotic arm, and thus can lead to unfavorable motions when inverse kinematics is directly applied. A potential solution is the integration of motion planning algorithms for joint rotations, either to replace inverse kinematics entirely or to complement it in specific segments of the task. In this way, the policy can retain direct control when no obstacles are present, while relying on motion planning to avoid collisions or handle complex arm movements before resuming normal policy-driven execution. As a complementary approach, additional constraints on rotations could be introduced during training to improve robustness to undesired configurations.

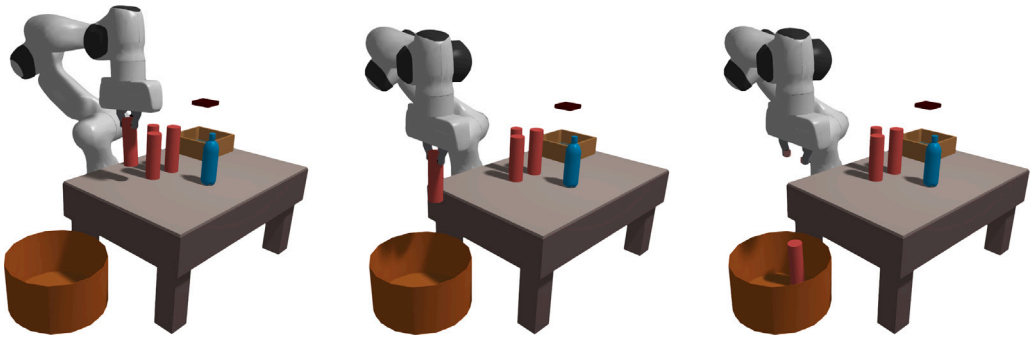


Fig. 10. Execution of cylinder removal during the clearing phase of the clear-and-place task. Cylinders are grasped from above (left), moved off the table and over the bin (center), and released (right).

Table 1
Success rates of the pick-and-place task for top grasping with different objects. The results are obtained by performing the task 10 times for 5 different seeds. The reported values are 95% confidence intervals for the grasp-and-lift phase (learned), the reaching phase for the place position (not learned), the place phase (learned), and the release phase (not learned).

Object	`pickUp`	`move`	`place`	`release`
Cylinder	100% ± 0.0	100% ± 0.0	100% ± 0.0	100% ± 0.0
Wine glass	100% ± 0.0	100% ± 0.0	100% ± 0.0	94% ± 7.0
Cube	100% ± 0.0	98% ± 3.5	92% ± 6.6	92% ± 6.6
Water bottle	86% ± 8.9	84% ± 7.0	64% ± 13.2	64% ± 13.2

Table 2
Success rates of the pick-and-place task for side grasping with different objects. The results are obtained by performing the task 10 times for 5 different seeds. The reported values are 95% confidence intervals for the grasp-and-lift phase (learned), the reaching phase for the place position (not learned), the place phase (learned), and the release phase (not learned).

Object	`pickUp`	`move`	`place`	`release`
Cylinder	92% ± 6.6	90% ± 7.8	84% ± 8.9	84% ± 8.9
Wine glass	94% ± 7.0	92% ± 10.3	90% ± 9.6	90% ± 9.6
Cube	94% ± 7.0	94% ± 7.0	84% ± 11.9	82% ± 10.3
Water bottle	100% ± 0.0	100% ± 0.0	92% ± 10.3	92% ± 10.3

5.4. Clear-and-place task

We now consider the *clear-and-place* task, which involves more complex operations. The objective is to pick up a target object from the table and place it inside a box. However, in this case, other objects on the table act as potential obstacles. Therefore, the robot must first remove all obstacles before grasping the target object and completing the task.

5.4.1. Simulation scenario

In the proposed scenario (see Fig. 8, right), we assume some cylinders are placed on the table, along with a bottle, and a box. The target object is the bottle, while the cylinders are obstacles. The designated area for the objects is divided into a grid, and the cylinders are randomly placed in a checkerboard pattern, excluding the row farthest from the robotic arm. The bottle, on the other hand, is randomly placed within the region extending from the last row of the grid to the end of the available placement area. The distracting cylinders must be removed from the table and transported to a designated location for disposal. Moreover, we assume that objects' removal is performed with a top grasp to avoid other cylinders, while the target object is taken with a lateral grasp.

5.4.2. Task specification

In terms of HTN planning, we introduce the task `clear-and-place (obj)`, which is decomposed into two main sub-tasks: `get(obj,side)` and `leave(obj,loc)`. While the latter remains the same as in the previous scenario, with `loc` as the disposal location, the former can be executed as a simple `take(obj,mode)` task if the target object is reachable either from the side or the front, with no additional action needed to remove cylinders. Otherwise, if obstacles are detected, the `take(obj,mode)` task is preceded by a preparatory `clear(obj,side)` sub-task, which iteratively removes blocking objects from the specified side according to the selected method. The clearing task is implemented by the recursive method `m_clear(obj,side)`, which terminates immediately if the object's side is free. Otherwise, if an obstacle `ob1` is present, a `remove(ob1)` task is invoked, followed by a recursive call to `clear(obj,side)` until the object becomes accessible. In this scenario, we assume that the `remove(obj)` task is performed using a top grasp, executed as `m_take_top(obj,top)`, followed by placement in the disposal location. The generated plan is therefore a sequence of pick-and-place operations, where the obstacles are first relocated outside the table, and then the target object is placed inside the box.

Table 3
Clear-and-place task success rate (successful pick-and-place of the bottle) and obstacles cleared rate, varying with the number of obstacles to clear. The reported values are 95% confidence intervals.

# Obstacles	General success rate	Obstacles cleared rate
1	98% ± 3.5	100% ± 0.0
2	94% ± 4.3	99% ± 1.8
3	94% ± 7.0	94% ± 5.4
4	94% ± 4.3	90% ± 3.4
5	82% ± 6.6	81% ± 2.1

5.4.3. Evaluation

In this scenario, we aim to assess the reliability of the generated plans that combine multiple pick-and-place operations in a cluttered environment. Specifically, we conducted 5 plan generation and execution tests with different numbers of randomly placed obstacles (from 1 to 5). As in the previous case, 50 runs have been performed for each situation (250 runs in total) using five different seeds.

While plan generation was consistently successful, plan execution posed greater challenges. In addition to measuring the overall task success rate — defined as successfully removing all cylinders and placing the final bottle in the box — we also evaluated the percentage of correctly removed cylinders. This metric is particularly relevant when multiple closely packed obstacles are present (e.g., four or more), as the gripper may unintentionally collide with other cylinders while attempting to grasp one. These collisions occur because the trained policy does not incorporate explicit spatial awareness of surrounding objects; obstacle proximity is only symbolically represented for planning. Thus, this metric provides insight into the policy's ability to grasp closely positioned objects without excessive collisions. The success rates are reported in Table 3.

The collected results highlight the effectiveness of the generated plans, which successfully integrate both trained and non-trained primitives. As expected, the plan execution success rate decreases as the number of obstacles increases, primarily due to the higher likelihood of colliding with the bottle while moving cylinders in cluttered environments. Additional collisions may also occur during the lifting phase if a cylinder is not grasped perpendicularly to the table. In real-world scenarios, these issues could be mitigated by incorporating obstacle avoidance methods during movement to prevent unintended contacts between the robot and objects. However, since obstacles were not present in the simplified training scenario, a potential improvement would be to retrain the policy with additional demonstrations to better constrain the picking and lifting motions. This would involve associating the retrained policy with new symbolic primitives and revised preconditions to address specific contexts (e.g., grasping in cluttered environments). This approach would ensure that the system can adapt to more complex scenarios while maintaining the modularity and reusability of the learned primitives.

5.5. Discussion

The proposed scenarios demonstrate the system in action, handling increasingly structured tasks that combine both trained and untrained primitives. We evaluated the transferability and compositionality of the learned primitives in a simulated environment that is more realistic than the one used during policy virtual demonstration and training. Within this setting, the HTN planning framework allows for the flexible integration of learned primitives into structured tasks, given a background knowledge and a repertory of already available operators and methods.

The proposed results on task composition and execution highlight the potential of the framework to incrementally learn and flexibly compose structured manipulation tasks across different environments and platforms. In particular, the successful generation and execution of multi-step plans in cluttered settings within a realistic simulator demonstrate the system's capacity to integrate both learned and non-learned components, while preserving adaptability and transferability. At the same time, the observed execution challenges (e.g., cylinder collision in cluttered situations) emphasize the need for incremental adaptation through the refinement and composition of sub-task policies as task complexity increases. A related issue stems from the direct use of inverse kinematics to track the gripper movements in redundant manipulators, which may lead to undesired configurations. A natural remedy is the integration of motion planning techniques, which can track the desired end-effector motion while ensuring feasibility and collision avoidance. On the other hand, the integration of learned skills into the HTN planning domain requires appropriate symbolic representation, including the definition of preconditions, effects, and associated methods. In the current work, we assumed these representations to be provided as specifications of learned skills, focusing instead on demonstrating structured plan generation and execution. As a first approach, it is possible to assume that trained policies are the implementation and specialization of abstract, already represented, primitive operators, possibly refined by additional policy-specific preconditions, provided as abstract descriptions of the additional contextual constraints needed by the trained policy. While this symbolic representation step is currently performed manually, it allows for precise and robust grounding of learned behaviors within the planner's symbolic model, ensuring reliable task composition and execution. The automatic derivation of symbolic action models from demonstrations or policy executions has been addressed in various frameworks [26,31,32], but it falls outside the scope of this work. Nonetheless, the modular and incremental design of our framework ensures compatibility with future advances in automatic symbol grounding, positioning it as a practical and extensible approach for combining low-level learning with high-level planning in the HTN planning framework.

6. Conclusion

In this work, we proposed a framework for incremental learning and flexible composition of structured robotic manipulation tasks. The approach combines imitation learning from virtual demonstrations with reinforcement learning to incrementally train reusable and transferable manipulation policies. These skills are collected in a structured repository and represented symbolically to support their reuse and adaptive composition via a Hierarchical Task Network (HTN) planning framework, enabling the generation and execution of long-horizon tasks.

As a specific feature of the proposed approach, learned primitives are trained in simplified virtual environments, using a limited number of demonstrations and under minimal assumptions about the manipulated objects and the robotic platform. This abstraction enables modular skill reuse and is designed to facilitate task transferability, by decoupling policy learning from robot-specific dependencies. We demonstrated the reusability, composability, and transferability of the learned skills in testing environments that are more realistic and complex than those used during primitive training. Experimental evaluations show high success rates across grasping, lifting, placing, and multi-step task execution scenarios.

As future work, we plan to extend the framework to real-robot deployment, building on the generalization already shown in sim-to-sim transfer across simulators. We also aim to explore the integration of motion planning and kinodynamic constraints to better handle collisions and respect the robot's kinematic and dynamic limits, particularly in cluttered and challenging manipulator scenarios. In parallel, we plan to extend the framework to more complex and heterogeneous manipulation problems, across different environments and platforms. We further plan to investigate the automatic synthesis of symbolic specifications for learned primitives. In particular, we envision leveraging generative AI tools to support the formulation and refinement of symbolic operator and method descriptions through user-in-the-loop mechanisms. This would extend the virtual demonstration process to include the interactive construction and validation of symbolic representations, fostering a more integrated, human-guided abstraction process that enhances both the transparency and adaptability of the planning domain.

CRediT authorship contribution statement

Giuseppe Rauso: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Riccardo Caccavale:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Alberto Finzi:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Giuseppe Rauso, Riccardo Caccavale, Alberto Finzi report financial support was provided by European Commission. Giuseppe Rauso, Riccardo Caccavale, Alberto Finzi reports financial support was provided by Italian Space Agency. Giuseppe Rauso, Riccardo Caccavale, Alberto Finzi report financial support was provided by Italian Ministry of University and Research (MUR). If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the projects: EU Horizon INVERSE (grant 101136067) and euROBIN (grant 101070596); Melody (PRIN PNRR CUP E53D23017550001); SPACE IT UP (ASI MUR Contract 2024-5-E.0 - CUP I53D24000060005).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2025.105274.

Data availability

Data will be made available on request.

References

[1] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, S. Levine, Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, 2017, ArXiv, abs/1709.10087.
[2] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N.M.O. Heess, T. Rothörl, T. Lampe, M.A. Riedmiller, Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, 2017, ArXiv, abs/1707.08817.
[3] S. Ross, D. Bagnell, Efficient reductions for imitation learning, in: Proc. of AISTATS, in: PMLR, vol. 9, 2010, pp. 661–668.
[4] O. Kroemer, S. Niekum, G. Konidaris, A review of robot learning for manipulation: Challenges, representations, and algorithms, J. Mach. Learn. Res. 22 (30) (2019) 1–82.
[5] H. Ravichandar, A.S. Polydoros, S. Chernova, A. Billard, Recent advances in robot learning from demonstration, Annu. Rev. Control. Robot. Auton. Syst. 3 (2020) 297–330.
[6] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, S. Levine, Composable deep reinforcement learning for robotic manipulation, in: Proc. of ICRA, 2018, pp. 6244–6251.
[7] S. Cheng, D. Xu, LEAGUE: Guided skill learning and abstraction for long-horizon manipulation, IEEE Robot. Autom. Lett. 8 (10) (2023) 6451–6458.
[8] B. Wu, R. Martín-Martín, L. Fei-Fei, M-EMBER: Tackling long-horizon mobile manipulation via factorized domain transfer, in: Proc. of ICRA, 2023, pp. 11690–11697.
[9] B. Abbatematteo, E. Rosen, S. Thompson, T. Akbulut, S. Rammohan, G. Konidaris, Composable interaction primitives: A structured policy class for efficiently learning sustained-contact manipulation skills, in: Proc. of ICRA, 2024, pp. 7522–7529.
[10] D. Kawakami, R. Ishikawa, M. Roxas, Y. Sato, T. Oishi, Learning 6DoF grasping using reward-consistent demonstration, 2021, ArXiv, abs/2103.12321.
[11] S. Song, A. Zeng, J. Lee, T. Funkhouser, Grasping in the wild: Learning 6DoF closed-loop grasping from low-cost demonstrations, IEEE Robot. Autom. Lett. 5 (3) (2020) 4978–4985.
[12] G. Rauso, R. Caccavale, A. Finzi, Incremental learning of robotic manipulation tasks through virtual reality demonstrations, in: Proc. of IROS, 2024, pp. 5176–5181.
[13] D.S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, F. Yaman, SHOP2: An HTN planning system, J. Artificial Intelligence Res. 20 (1) (2003) 379–404.
[14] P. Sharma, D. Pathak, A.K. Gupta, Third-person visual imitation learning via decoupled hierarchical controller, in: Proc. of NIPS, 2019, pp. 2597–2607.
[15] P. Sermanet, C. Lynch, J. Hsu, S. Levine, Time-contrastive networks: Self-supervised learning from multi-view observation, in: Proc. of CVPRW, 2017, pp. 486–487.
[16] A. Handa, K. Van Wyk, W. Yang, J. Liang, Y.-W. Chao, Q. Wan, S. Birchfield, N. Ratliff, D. Fox, DexPilot: Vision-based teleoperation of dexterous robotic hand-arm system, in: Proc. of ICRA, 2020, pp. 9164–9170.
[17] P. Wu, Y. Shentu, Z. Yi, X. Lin, P. Abbeel, GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators, in: Proc. of IROS, 2024, pp. 12156–12163.
[18] P. Sharma, L. Mohan, L. Pinto, A. Gupta, Multiple interactions made easy (MIME): Large scale demonstrations data for imitation, in: Proc. of CoRL, 2018, pp. 906–915.
[19] S.P. Arunachalam, S. Silwal, B. Evans, L. Pinto, Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation, in: Proc. of ICRA, 2023, pp. 5954–5961.
[20] R. Caccavale, M. Saveriano, A. Finzi, D. Lee, Kinesthetic teaching and attentional supervision of structured tasks in human-robot interaction, Auton. Robots 43 (6) (2019) 1291–1307.
[21] R. Caccavale, M. Saveriano, G.A. Fontanelli, F. Ficuciello, D. Lee, A. Finzi, Imitation learning and attentional supervision of dual-arm structured tasks, in: Proc. of ICDL-EpiRob, 2017, pp. 66–71.
[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, ArXiv, abs/1707.06347.
[23] J. Ho, S. Ermon, Generative adversarial imitation learning, in: Proc. of NIPS, 2016, pp. 4572–4580.
[24] S. Nasiriany, H. Liu, Y. Zhu, Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks, in: Proc. of ICRA, 2022, pp. 7477–7484.
[25] M.J. McDonald, D. Hadfield-Menell, Guided imitation of task and motion planning, in: A. Faust, D. Hsu, G. Neumann (Eds.), Proc. of CoRL, Vol. 164, PMLR, 2022, pp. 630–640.
[26] A.M. Zanchettin, End-to-end action model learning from demonstration in collaborative robotics, Robot. Auton. Syst. 193 (2025) 105071.
[27] M. Mayr, F. Ahmad, K. Chatzilygeroudis, L. Nardi, V. Krueger, Skill-based multi-objective reinforcement learning of industrial robot tasks with planning and knowledge integration, in: Proc. of ROBIO, 2022, pp. 1995–2002.
[28] A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, D. Lange, Unity: A general platform for intelligent agents, 2018, ArXiv, abs/1809.02627.
[29] O. Taheri, N. Ghorbani, M.J. Black, D. Tzionas, GRAB: A dataset of whole-body human grasping of objects, in: Proc. of ECCV, Springer International Publishing, 2020, pp. 581–600.
[30] S. Brahmbhatt, C. Ham, C.C. Kemp, J. Hays, ContactDB: Analyzing and predicting grasp contact via thermal imaging, 2019, ArXiv, abs/1904.06830.
[31] A. Arora, H. Fiorino, D. Pellier, M.M.. Etivier, S. Pesty, A review of learning planning action models, Knowl. Eng. Rev. 33 (2018).
[32] T. Silver, R. Chitnis, J. Tenenbaum, L.P. Kaelbling, T. Lozano-Pérez, Learning symbolic operators for task and motion planning, in: Proc. of IROS, 2021, pp. 3182–3189.