Learning to Rearrange Objects in Confined Environments

Alessandro Palleschi*, Marion Lepert+, Wenzhao Lian#, Lucia Pallottino*, Jeannette Bohg+

***Abstract*— Robots capable of rearranging objects in cluttered and confined spaces have numerous real-world applications, such as in retail, logistics, and household tasks. However, environmental constraints and visual occlusions make it difficult to predict the effects of robot-environment interactions, presenting a significant challenge for rearrangement planning. Existing solutions rely on simplified assumptions, such as full observability, and typically use collision-free, single-object prehensile manipulation strategies that are less effective in partially observable settings. To address these limitations, this paper proposes a data-driven approach that leverages deep reinforcement learning to learn a rearrangement policy that combines two actions: pushing and pick-and-place. Specifically, the approach uses fully convolutional networks and Q-learning to make dense pixel-wise predictions of expected rewards for the two actions from side-views of the cluttered and confined space. At each step, the learned policy executes the action with the highest Q-value given the current observation. The proposed method is evaluated in simulation, where a learned policy is rolled out to rearrange up to 14 objects inside a confined cabinet. Results show that the approach achieves an average success rate improvement of 31.7% compared to baselines. Overall, this work demonstrates the efficacy of a data-driven approach to enable robots to effectively rearrange objects in complex, cluttered environments.**

I. Introduction

Enabling robots to operate efficiently in everyday human environments has been a longstanding goal of robotics research. However, unlike the structured and controlled industrial environments where robots have traditionally been deployed, human environments are often unstructured and variable. In particular, they are typically cluttered and constrained, and require robots to interact with and rearrange objects in the environment to accomplish specific tasks. As such, developing effective strategies for object rearrangement in cluttered and confined environments has become an increasingly important area of research, with applications in fields ranging from retail and logistics to household tasks.

In this work, we focus on the multi-object sorting problem where the goal is to manipulate and rearrange objects in a confined environment such that objects of one category are grouped together. Many works have addressed the problem of manipulation planning in confined spaces [1]–[5], but most of them focus on searching for or retrieving a target object. Thus, they are not concerned with the final arrangement of the other objects in the environment at the end of the task. The problem we consider adds more planning complexity since it represents a long-horizon manipulation problem that requires manipulating all the objects to reach a specified goal state. In addition, the presence of occlusions caused by clutter and uncertainty about the real state of the environment increase the complexity of this planning problem that is in general $\mathcal{NP}$-hard [6].

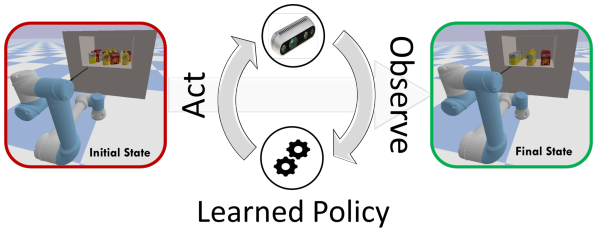


Fig. 1: Schematic representation of the rearrangement problem. Given a visual observation of a set of objects in a confined workspace, the robot uses a learned policy to select actions that bring the environment into a sorted state.

Existing solutions for this problem have been developed under strong assumptions, such as complete observability and known object models, relying only on single-object collision-free prehensile manipulations. However, robots working in real-world settings have to deal with partially observable and uncertain environments, potentially interacting with objects never seen before and with levels of clutter that might hinder completely avoiding contact with other objects. Strategies that utilize contact-rich manipulation techniques, such as pushing, which allow for concurrent multi-object interaction, have the potential to be more efficient than strategies limited to single-object manipulation [7]. Nonetheless, planning a sequence of manipulation actions over a long horizon in partially observable and contact-rich environments is challenging due to the complexity of accurately predicting the effects of an action. Uncertainty about physical object parameters and the state of the world makes an open-loop execution of long sequences of actions practically impossible.

To overcome these challenges, this work proposes to leverage model-free deep reinforcement learning to learn a rearrangement policy based solely on visual inputs. Model-free approaches represent a good solution for reactive decision-making in uncertain and cluttered environments, where it might be complex and costly to derive an internal model for forward planning, since physics in

Toyota Research Institute provided funds to support this work. This work has also received funding from Intrinsic, the European Union Horizon 2020 research and innovation program under agreement no. 871237 (SOPHIA), and the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab and FoReLab projects (Departments of Excellence).

* Research Center E. Piaggio, Department of Information Engeneering, University of Pisa alessandro.palleschi@phd.unipi.it, lucia.pallottino@unipi.it

+ Stanford University, Department of Computer Science {lepertm, bohg}@stanford.edu

Intrinsic Innovation LLC in CA, USA. wenzhaol@intrinsic.ai

interaction-rich environments can easily evolve into multi-modal and non-smooth distributions [3]. In this framework, the robot can interact with the environment using two action primitives, pushing and pick-and-place, and learns rearrangement policies that combine these two actions by trial-and-error in a deep Q-learning framework. Therefore, the agent is not limited to collision-free prehensile actions. Given a partially occluded observation of the confined environment, the rearrangement policy uses the Q-maps computed by fully convolutional neural networks to select and execute the action with the highest Q value at each time step. We evaluate the performance of the proposed approach in simulated cluttered environments with up to 14 objects. We show that the learned policy allows the robot to effectively and efficiently operate in highly cluttered and partially observable settings, where heuristic-based baselines are not effective, and demonstrate that combining pushing and pick-and-place actions increase efficiency and performance of the learned policy.

The main contribution of this paper is to frame multi-object sorting with partial observability as an end-to-end reinforcement learning problem, where the robot learns how to combine pushing and pick-and-place actions to sort all the objects based on lateral RGB-D views of the confined environment. In addition, to allow learning policies transferable to large varieties of objects with different geometry and appearance, we propose to describe the state of the world, i.e., the features used by the network to evaluate the usefulness of an action, through an abstract image-based representation that highlights task-relevant information, such as, class memberships of the observed objects and position of the target sorting regions.

II. Related Works

A. Manipulation in confined spaces

Many works focus on grasping objects from cluttered environments [1], [2], [5], [8], [9] or on searching and retrieving objects in confined spaces [3], [4], [10]–[13]. Even though all these works are applied to confined and cluttered environments and require planning actions to rearrange objects within the scene, the final arrangement of the objects is not essential for defining the task's success. In contrast, in our setup, the poses of all the objects are relevant as the goal is to sort the objects to reach a specific arrangement. In addition, most prior approaches reduce the complexity of the problem by assuming known object models and poses [2], [5], [10], [13], and restrict manipulation to one object at a time to avoid object-object interactions that can be hard to model [4], [10]–[12].

B. Object rearrangement

The problem addressed in this work is a particular instance of the object rearrangement problem. The rearrangement problem has been described as a "canonical task" for evaluating embodied AI [14], and can be characterized by the complexity of the considered environment (level of clutter and containment), the set of actions available to the robotic platform, and the level of generalization to novel environments and objects. The problem of rearranging/sorting objects using both prehensile and non-prehensile actions has been widely investigated for planar/tabletop scenarios [7], [15]–[18].

The approach presented in [18] considers long-horizon rearrangement planning tasks in tabletop scenarios, where multiple (unknown) objects have to be rearranged to a specific goal configuration (given in the form of a goal image). It relies on segmented visual data as input, and its performance heavily depends on the quality of the segmentation. In addition, it limits the robot to use only prehensile pick-and-place actions to reach the desired arrangement.

Other methods focus instead on large-scale, multi-object rearrangement, where planar pushing actions are used to move multiple objects at the same time to increase efficiency. The approach proposed in [15] exploits iterative local search and assumes the pusher can move in and out of the pushing plane at any location. Other works use learning-based Monte Carlo Tree Search for problems where the pusher's motion is constrained to the pushing plane and the goal configuration is not explicitly given [16], or exploit a kinodynamic planning framework with dynamic planning horizons [7]. Pan and Hauser [17] propose a different solution for planning large-scale object sorting, combining overhead grasping and planar pushing. They use an analytical method that provides completeness guarantees under the assumptions that objects are not pushed to regions not reachable by the robot arm and that the state is fully observable.

All these methods are restricted to planar workspaces where the gripper can reach the objects from the top, and many assume full observability. Their applicability to confined environments, where the robot can only access and observe the workspace from the side, is limited. Environmental constraints reduce the accessibility for some objects, basically making collision-free manipulation impossible, and occlusions due to clutter introduce uncertainty about the true state of the environment. In addition, most of the proposed methods use a single action, either grasping or pushing, to perform the task. When multiple actions are used, the pushing action is mostly used to enable future grasps and not as a rearrangement action.

Regarding the approaches developed for object rearrangement in confined spaces, they generally exploit policies based on collision-free prehensile actions [19]–[21]. These methods are able to solve both monotone (where all objects need to be picked only once) and non-monotone (where objects need to be picked multiple times and relocated inside the confined environments) scenarios. However, they assume having access to the pose of every object, i.e., full observability. Moreover, they are often limited to specific objects, e.g., uniform cylinders with known dimensions, and the rearrangement policies are designed to avoid object-object interactions. This latter constraint would lead to infeasible solutions in many cases where the clutter could prevent collision-free trajectories.

(a) Confined space (b) Robotic arm

Fig. 2: Setup: (a) cubic workspace with N objects inside, (b) the robot equipped with a pushing blade with a suction cup at the end of the tool.

III. Problem Statement

Consider a cubic workspace $\mathcal{W} \subset \mathbb{R}^3$ containing N movable objects $\mathcal{O} = \{o_1, \ldots, o_N\}$ (see Fig. 2a), and a robot arm placed in front of the cubic workspace. The robot arm can observe and access the workspace only from one side (lateral access). To facilitate manipulation in confined and narrow environments, the robot is equipped with a pushing blade with a suction cup at the end of the tool, see Fig. 2b. The robot can interact with the environment using two parametrized actions: linear pushing using the blade and pick-and-place through activation and deactivation of the suction cup. The set of parametrized actions is indicated with $\mathcal{A}_\rho = \{\mathcal{T}(\rho_\mathcal{T}), \mathcal{P}(\rho_\mathcal{P})\}$, where $\mathcal{T}$ is a pick-and-place action, $\mathcal{P}$ is a linear pushing action, and ρ_i represents the parameters for an action i. The objects inside the workspace are partitioned into a finite number of k classes. Each of these classes has been assigned to a specific region inside the workspace $\mathcal{W}_k \subset \mathcal{W}$. A sorted state is when the objects of each class k are inside the target region $\mathcal{W}_k$.

Given these definitions, it is possible to state the following rearrangement problem: given an initial unsorted configuration for the objects $\mathcal{O}$ inside the workspace $\mathcal{W}$, the robot should plan and execute a sequence of pushing and pick-and-place actions that brings the environment to a sorted state using observations from a fixed camera placed in front of the shelf.

IV. Proposed Approach

We frame this sorting problem as a Markov Decision Process (MDP): given a state s_t at time t the agent selects an action $a_t \in \mathcal{A}_\rho$ to execute according to a policy $\pi(s_t)$. As a result of this action, the system transitions to the new state s_{t+1}, and the agent receives an immediate reward $R(s_{t+1})$ based on the results of that action. The goal is to find an optimal policy π^* that maximizes the discounted sum of future rewards. We use model-free double deep Q-learning [22] to learn a greedy deterministic policy that, given a state s_t, selects the action a_t that maximizes the action-value function $Q_\pi(s_t, a_t)$.

A diagram describing the proposed architecture is shown in Fig.3. The presented approach uses fully-convolutional neural networks (FCNs) to model Q-functions that estimate the expected reward for each action candidate given the current observation o_t, which represents the state s_t.

A. State Representation

In the proposed approach, the state s_t comes in the form of a visual observation o_t that is a visual representation of the inside of the cabinet annotated with task-relevant information. This observation is obtained by processing the images recorded from a fixed-mount camera observing the cabinet from the side. As discussed in other works [3], [16], [23], providing the network with an abstract representation rather than raw images can help learn policies that generalize to different objects and settings. In fact, policies can learn to exploit task-relevant information and features, such as the object classes and the specific target region, omitting details that are irrelevant to the task, such as background color, lighting sources, and item texture.

In our approach, we feed the FCNs with both depth and semantically annotated RGB data. The robot first captures an RGB-D image from the front of the cabinet. By converting this side-view image into a point cloud and projecting it onto the horizontal plane, we create a heightmap or top-down view of the inside of the cabinet (Fig. 3). The margins of the map are preset to the dimensions of the cabinet, which is assumed known a priori.

We assume that instance segmentation is applied to the RGB image to detect and obtain the masks and classes for all the visible objects and object parts. Instance segmentation can be performed using state-of-the-art perception algorithms such as [24], [25], which tackle unknown objects. In our visual heightmap representation (Fig. 3), objects of the same class share the same color, regardless of their real appearance. The free and occluded spaces are represented by white and grey-colored areas, respectively. Additionally, visible sections of the cabinet corresponding to a target location are colored based on the class of the object they are meant to store. Note that we are assuming the perception returns masks only for the visible regions of the objects. We are not assuming to have access to amodal perception to infer the entire geometry of occluded objects from partial views [26], and no information is available for objects completely occluded. The semantic top-down view in the form of an RGB image is then concatenated with the single channel depth image, where each pixel contains the height-from-bottom value of the heightmap, to obtain the full visual observation $o_t \in \mathbb{R}^{\mathrm{H} \times \mathrm{W} \times 4}$ that forms the input to the FCNs.

B. Primitive Actions

The action set $\mathcal{A}_\rho$ is modeled using two parametrized motion primitives, $\mathcal{T}(\rho_\mathcal{T})$ and $\mathcal{P}(\rho_\mathcal{P})$, where $\mathcal{T}$ is a pick-and-place action, $\mathcal{P}$ is a linear pushing action, and ρ_i represents the parameters for an action i.

In the following, we provide details on the action parameters, and on the architecture of the FCNs that take as input the visual observation o_t and select the action parameters.

1) Pushing:

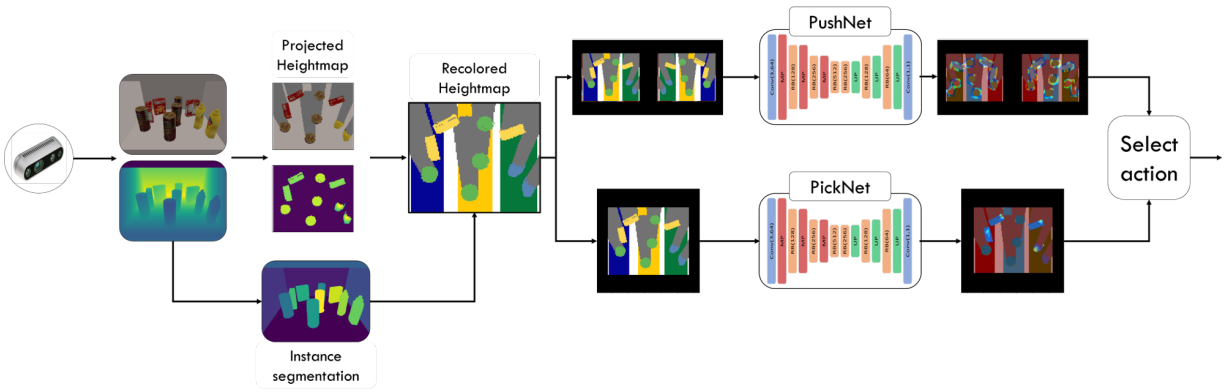


Fig. 3: Proposed architecture: Given a side view of the environment acquired through an RGB-D sensor, we first transform the visual observation of the inside of the cabinet to create a recolored orthographic view, highlighting the class membership of each visible object and the area where the objects should be sorted. The heightmap is sent to two FCNs that generate pixel-wise Q-maps corresponding to the expected rewards for taking a push or a pick-and-place action at each specific pixel.

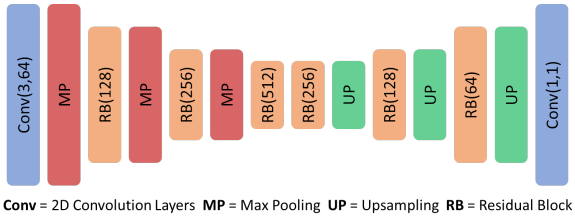


Fig. 4: Architecture of the fully convolutional residual network used to compute the Q-maps from the current observation. Conv(k,i) denotes a convolutional layer with $k \times k$ filters and i channels, RB(i) represents a residual block composed of two convolutional layers with 3×3 filters and i channels.

a) Primitive definition: To model the pushing primitive $\mathcal{P}$, we follow Zeng et al. [27]. It is defined by $\rho_{\mathcal{P}} = (x_{\mathcal{P}}, y_{\mathcal{P}}, \theta_{\mathcal{P}})$. Here, $(x_{\mathcal{P}}, y_{\mathcal{P}})$ are the 2D coordinates of the starting position of the tip of the pushing blade, expressed relative to a local frame with the z-axis being normal to the ground plane, the x-axis aligned with the cabinet's depth dimension and the y-axis aligned with the cabinet's width dimension. $\theta_{\mathcal{P}}$ specifies the direction of the pushing action also with respect to this local frame.

When executing this action, the robot first reaches the point $(0, y_{\mathcal{P}})$ in front of the cabinet. Then it linearly approaches the starting point inside the cabinet at the specific depth $x_{\mathcal{P}}$ and executes a straight pushing action of length $l_p = 8$cm at a constant distance of $z_p = 5$cm from the bottom of the cabinet to reduce the risk of toppling. This straight line push is in the direction specified by $\theta_{\mathcal{P}}$. Afterward, it retracts from the cabinet by reversing this trajectory.

b) Push-Net: The goal of the pushing network is to derive the parameters of action ρ_p from the current observation o_t. The pushing network is composed of an initial 7-layer fully convolutional residual network, interleaved with three layers of spatial 2×2 max-pooling, that takes as input the current observation and outputs a spatial feature map. This map is fed as input to a second 7-layer fully convolutional residual network (with three layers of spatial bilinear $2\times$ upsampling) that outputs a dense pixel-wise map of Q values with the same size as that of the original image (see Fig 4).

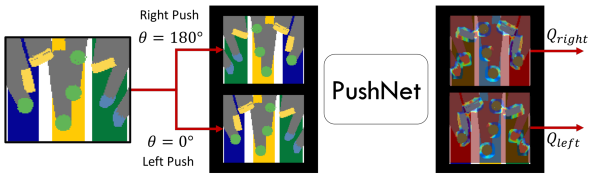


Fig. 5: Different pushing directions are handled by rotating the observation fed to the network. In this work we model two pushing directions (left and right) corresponding to a $0\,\text{deg}$ and a $180\,\text{deg}$ rotation.

As done in previous works [27], [28], different pushing directions are handled by rotating the input image sent to the network to obtain the corresponding Q-maps. This work considers two directions modeling left and right pushing actions, corresponding to a $0°$ and a $180°$ rotation of the input image, respectively (see Figure 5). Rotations are done with nearest-neighbor sampling to avoid blurring artifacts, as well as 0-padding to maintain fixed image sizes.

The pixel with the highest predicted probability among the two maps determines the parameters $\rho^*_{\mathcal{P}} = (x^*, y^*, \theta^*)$ for the pushing primitive to be executed: the 2D location of a pixel determines the pushing starting position, and the orientation of the map determines the pushing direction.

2) Pick-and-Place:

a) Primitive definition: The action is specified by a pair of 2D picking and placing poses $\mathcal{T} = \{\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}\}$. Each pose is parameterized by a vector $\rho_{\text{i}} = (x_{\text{i}}, y_{\text{i}})$ describing the 2D positions of the tip of the pushing blade (equipped with a suction cup) at the picking and releasing poses. These poses are expressed relative to the same local frame as the push primitive.

When executing this action, the robot first reaches the point $(0, y_{\text{pick}})$ in front of the cabinet, and then linearly approaches the picking point inside the cabinet at the specific depth x_{pick}. Once the point is reached, the suction cup

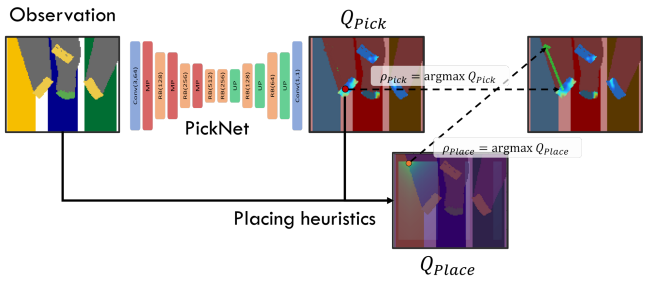


Fig. 6: Procedure used to select the parameters of the pick-and-place primitive. An FCN is used to determine the picking pose from the current observation. The placement pose is instead computed using a heuristic function based on the class of the picked object and associated target region, as well as the current state of the environment.

is activated if contact is detected, and the robot linearly retracts from the cabinet and eventually reaches a specified home configuration. If an object has been grasped, the robot executes the motion to reach the placement position with the same strategy.

b) Pick-Net: This network takes as input the visual observation at time t and predicts the parameters $\rho_{\mathcal{T}} = ((x_{\text{pick}}, y_{\text{pick}}), (x_{\text{place}}, y_{\text{place}})) = (\rho_{\text{pick}}, \rho_{\text{place}})$ for the pick and place action $\mathcal{T}$. We use a two-level approach: an FCN with the structure shown in Fig. 4 outputs a dense pixel-wise map of Q values of the same size as the original image. The pixel with the highest value is used to select the parameters ρ_{pick} of the picking primitives. The placing position is then selected using a heuristic that depends on the class of the grasped object (obtained from the color of the pixel selected by the network) and the level of clutter. A placement pose will be towards the back of the cabinet and close to the center of the object's target region, and does not obstruct objects of different classes. An example is shown in Fig. 6.

C. Action Selection

At each time t, the two networks take as input the current observation o_t and return three Q-maps $Q(o_t, a_\rho)$ corresponding to pick-and-place, left and right pushing, respectively. The policy selects the action and corresponding parameters with the highest Q value:

$$\pi(o_t) = \underset{a_\rho \in \mathcal{A}_\rho}{\text{argmax}}\, Q(o_t, a_\rho). \tag{1}$$

The action is executed by the robot and, after the execution, it observes the scene again and the process is repeated until the task is completed.

D. Training Details

1) Reward: The reward function is not overly shaped to avoid predetermining the robot's behavior:

$$R_{\mathcal{T}}(s_{t+1}) = \begin{cases} -10 & \text{if } s_{t+1} \notin \mathcal{V}, \\ 0 & \text{if } s_{t+1} \in \mathcal{G}, \\ -1 & \text{otherwise}, \end{cases} \tag{2}$$

where we indicate with $\mathcal{G}$ and $\mathcal{V}$ the sets of goal states and valid states, respectively. A generic valid state is defined as a state where all the objects are placed stably inside the confined environments. A constant negative reward of -1 is given for actions that do not immediately lead to a goal state, a 0 reward for the action that solves the task, and an additional negative reward of -10 if the action brings the system into a non-valid state (e.g., some objects are dropped off the cabinet).

2) Loss Function: Similar to [27], we train the two networks using the Huber loss function

$$\mathcal{L}_i = \begin{cases} \frac{1}{2}\delta_i^2, & \text{if } |\delta_i| < 1 \\ |\delta_i| - \frac{1}{2}, & \text{otherwise}, \end{cases} \tag{3}$$

to minimize the temporal difference $\delta_i = Q(s_i, a_i; \theta_{a_i}) - y_i$, where

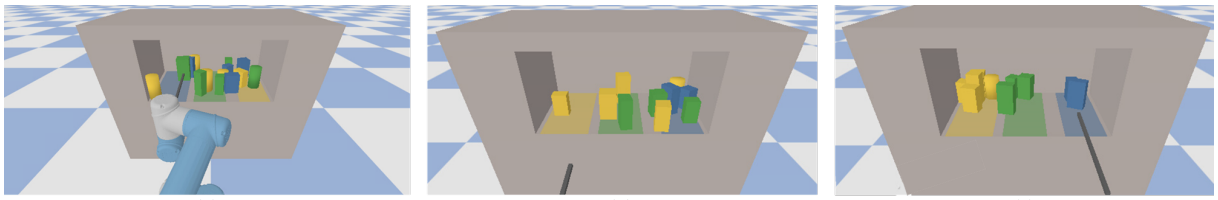
$$y_i = R_{a_i}(s_{i+1}) + \gamma Q(s_{i+1}, \underset{a'}{\text{argmax}}(Q(s_{i+1}, a'; \theta_{a'})); \theta'_{a_i}).$$

We use a double deep Q-learning framework to estimate the Q-function, where θ_{a_i} and θ'_{a_i} are the parameters of the primary FCN and the target network parameters for action a at iteration i, respectively. The gradients are passed only through the single pixel $\mathbf{p}$ and through the network from which the value for action a_i is computed. All other pixels at iteration i backpropagate with 0 loss. At each training iteration, the weights of the target network are updated through Polyak averaging with rate $\beta = 10^{-3}$: $\theta'_{a_i} \leftarrow (1-\beta)\theta'_{a_i} + \beta\theta_{a_i}$.

3) Data collection and training: We collect data in Pybullet with a UR5 robot equipped with a pushing blade (see Fig.7(a)). At the beginning of each episode, N objects are randomly placed inside a simulated cabinet of dimensions $40\text{cm} \times 60\text{cm} \times 27\text{cm}$, as in Fig.7(b). The objects are cuboids and cylinders of random sizes. Each object is randomly assigned one of three classes, and each class has a randomly located target region.

A simulated camera acquires the observations and the segmentation masks of the visible objects. We use green, yellow, and blue pixels to represent the membership of each object to one of the three classes in the recolored observation input of the network. An episode ends when: 1) all the objects are correctly sorted (Fig.7(c)); 2) an object falls out of the cabinet; 3) the robot exceeds a maximum number of actions. The collected observations, actions, and rewards are saved into an experience replay buffer. During training, a batch of size 32 is randomly sampled from the replay buffer for training.

The networks are trained in PyTorch [29] with an NVIDIA Quadro P5000 on an Intel Xeon CPU E5-2643 v4 clocked at 3.40GHz. We use the Adam optimizer with a fixed learning rate of 10^{-4} and weight decay of $2 \cdot 10^{-5}$. In this work, we employed a constant future discount $\gamma = 0.9$. The learning algorithm uses RL with a curriculum [30] that gradually increases the number N of objects in the scene. We start the learning with $N = 1$, and when the average success rate over the last 50 episodes is over 90%, we increase N by one, up to a maximum of 8 objects in the scene. For each stage of



(a) (b) (c)

Fig. 7: (a) A UR5 equipped with a pushing blade interacting with a simulated confined environment. (b) Example of an initial arrangement for 10 objects and three different classes. Objects of the same color belong to the same class. The target regions are shown only for clarity. (c) The robot manipulates the objects to bring them inside the assigned target regions.

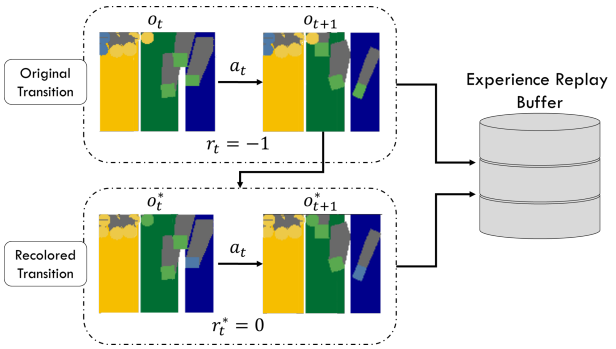


Fig. 8: Hindsight Experience Replay: during training, the robot executes an action a_t based on the current observation o_t, bringing the environment into a new state, corresponding to observation o_{t+1}. In this state, all the objects are inside the three target regions but are not correctly sorted. Thus, the misplaced objects are reassigned to a different class, obtaining a new transition with two recolored observations (o_t^* and o_{t+1}^*) and modified reward $r_t^* = 0$. Both the original transition and the relabeled one are inserted into the experience replay buffer.

the curriculum, we use an ϵ-greedy exploration policy with ϵ starting from 0.8 and then annealed over training to 0.01.

In addition, we use the hindsight experience replay (HER) to deal with sparse rewards [31]. Indeed, while rearranging the objects, the robot may bring the system to a state where all the objects are inside the available target regions, but at least one object is not in its assigned target region. This condition would match a correctly sorted state if the classes of some objects were different. To make sure that the agent can learn from these situations as well, the objects are reassigned post hoc to a different class so that this final state corresponds to a sorted one and relabel the reward for the transition leading to this state, as shown in Figure 8. This single relabeled transition, composed of the recolored observation, the action, and the modified reward, is then included in the replay buffer for further training.

V. Experimental Evaluation and Results

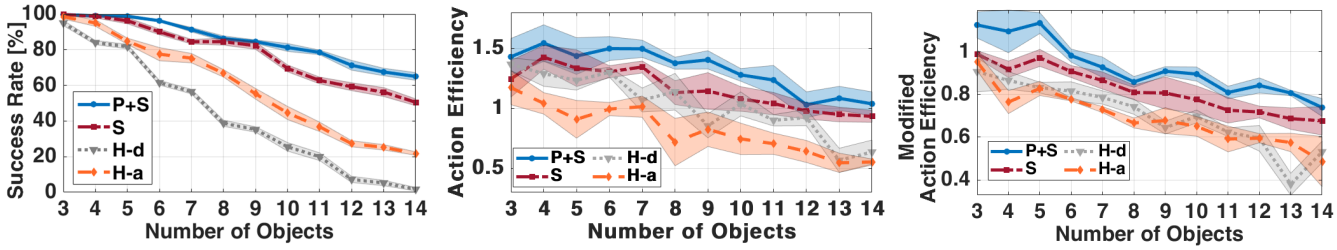
We evaluate the approach in simulation. The goal is to assess whether the proposed approach is able to let the robot learn efficient and nontrivial rearrangement strategies in cluttered scenarios, and to understand how the pushing actions influence the performance.

a) Baselines: To this end, the approach that uses both pushing and pick-and-place using the suction cup (labeled as **P+S** in the following) is compared with three baselines: **1)** a modified version of the proposed algorithm that uses only pick-and-place actions by means of the suction cup (named **S**) to solve the task; **2)** a heuristic-based method (named **H-d**), where the robot uses only pick-and-place actions, but the parameters of the picking action are computed using a simple heuristic that evaluates each pixel's distance from the target region and selects the pixel with the largest distance. **3)** a heuristic-based method (named **H-a**), where the robot exploits only pick-and-place actions, but uses a heuristic to select the object to pick based on the distance from the target region and the distance from the back of the cabinet. This baseline favors objects that are in the front of the cabinet, as they are generally more accessible and could potentially occlude other objects.

Fig. 9: Objects used at test time for the experimental evaluation of the algorithm.

b) Scenarios: We test these methods by executing a series of sorting tasks with different numbers of objects, from a minimum of 3 to a maximum of 14, that are randomly placed inside the confined environment. The objects used for the testing are a subset of the objects from the YCB dataset, shown in Figure 9. The objects are characterized by different shapes, texture, and dimensions, and have been chosen to show that the proposed architecture can effectively generalize to objects that vary in geometry and appearance, thanks to the abstract representation of the camera observation described in Section IV-A. For each number of objects N, we execute 100 runs. In each run, the N objects are sampled from the set shown in Fig.9 so that at most three different types are present.

c) Metrics: The method and the baselines are evaluated according to three metrics: **i)** the average % success rate (**SR**) over the 100 runs, where a success indicates that the agent has been able to sort all the objects with less than 40 actions; **ii)** the average action efficiency (**AE**) for successful



(a) Success Rate (b) Action Efficiency (c) Modified Action Efficiency

Fig. 10: Results with different numbers of YCB objects from the testing set in random initial arrangements for the three methods: **P+S** is our approach, **S** is a pick-and-place-only variant of our approach, and the two heuristic-based pick-and-place-only baselines (**H-d** and **H-a**). Metrics: (a) average % success rate (SR), (b) average action efficiency (AE) for successful runs: AE $= \frac{\#\text{objects}}{\#\text{actions until completion}}$, and (c) modified average action efficiency (mAE) for successful runs that considers the number of objects to sort in the initial the arrangement: mAE $= \frac{\#\text{objects to sort}}{\#\text{actions until completion}}$. The plots report mean and standard deviation computed over five different random seeds. For each number of objects N the policy has been rolled out for 100 random initial arrangements.

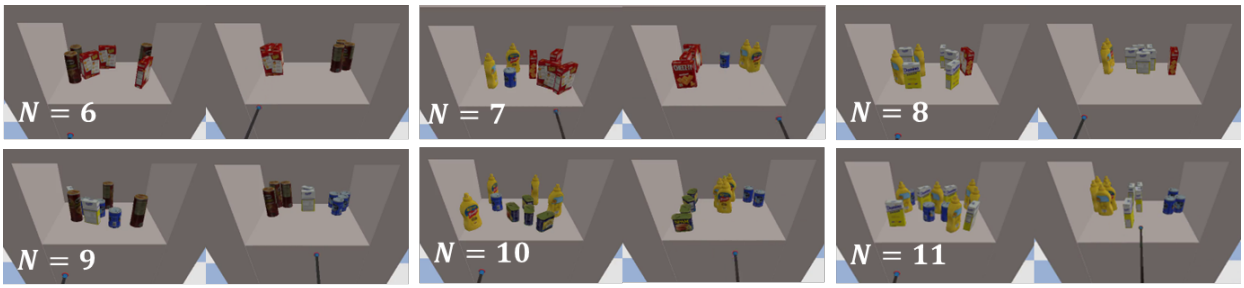


Fig. 11: Frames from the simulations for different number of YCB objects from the testing set in the scene. For each case, we show the initial configuration of the objects (left) and the final rearranged state (right).

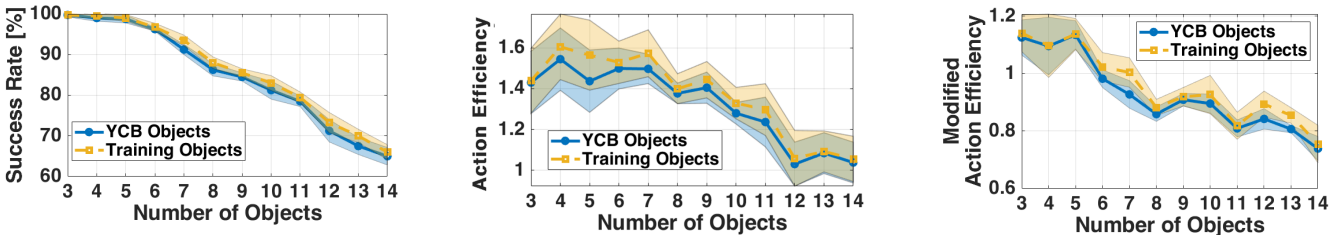
runs, measured as $\frac{\#\text{objects}}{\#\text{actions until completion}}$; **iii)** the average of a measure of the action efficiency modified to account for the initial arrangement of the objects (**mAE**), where the number of unsorted objects in the initial arrangement is considered instead of the total number of objects. For all the metrics, higher is better.

d) Results: The results are collected in Fig. 10, where, for the three metrics, we report the mean and standard deviation computed over 5 different random seeds. The simple heuristic-based baseline **H-d** is effective with up to five objects in the scene, but the performance drops drastically with any additional number of objects. The other heuristic-based baseline **H-a** performs better than **H-d**, in general, but the performance degrades with more than seven objects in the scene. In contrast, both **P+S** and **S** maintain good performance in more cluttered settings. This shows that, as expected, simple heuristics do not generalize well to complex environments. Our approach instead learns strategies that are effective even with levels of clutter not seen during training, since the robot has been trained with at most eight objects in the scene. However, the data reported in Fig.10(a) shows that the success rate for both **P+S** and **S** also degrades quite clearly as the number of objects increases – although never as low as the heuristic baselines.

The main failure case is due to objects falling out of the cabinet when the robot moves inside the confined environment. The more clutter, the more likely this is to happen. This often occurs when the robot retracts from the cabinet after picking an object. Contact between the picked object and other objects, or other indirect motion in the scene, might lead to some objects falling. Since every action is performed by the robot in open loop, the current framework is not designed to identify and properly react to these situations. These findings suggest that the introduction of an online mechanism for detecting undesired movements through, e.g., tactile sensing or visual feedback, as well as introducing proper reactive strategies, can improve performance in more cluttered environments.

It is worth noting that the proposed method with both pushing and pick-and-place actions outperforms the baseline using only pick-and-place. This seems to support the insight that including more complex non-prehensile actions is beneficial for this specific task. Regarding the efficiency of the solutions found (Fig.10(b)-(c)), both the two learning-based approaches perform better than the heuristic-based ones, with **P+S** being slightly more efficient than **S**. A possible explanation is that pushing actions allow moving multiple objects at the same time, potentially reducing the overall number of actions needed to reach a sorted state.

Eventually, Figure 12 shows the performance of the proposed algorithm **P+S** when the policy is rolled out in environments with objects sampled from the training set, i.e., simple cuboids and cylinders, compared to the performance when using the YCB objects shown in Fig. 9. Note that



(a) Success Rate (b) Action Efficiency (c) Modified Action Efficiency

Fig. 12: Comparison of the performance of the proposed approach in environments with objects sampled from the training set, cuboids and cylinders, or from the testing set composed of objects from the YCB dataset.

the algorithm does not experience a significant performance reductions, demonstrating that with our representation of the input the learned policy is able to generalize to objects not seen during training. However, it is worth noting that most objects from this YCB dataset are shaped like cylinders and cuboids.

VI. Conclusion

In this work, we presented a data-driven approach to derive rearrangement policies in confined spaces. We show that by modeling the task as an end-to-end model-free reinforcement learning problem, a robot can learn effective rearrangement strategies in cluttered and confined environments. Future work will evaluate the approach with real hardware to test the method's robustness to perception inaccuracies that can cause noisy segmentation or incorrect object classifications. Meanwhile, we will extend the method by learning directly from experience the pushing length and the placement pose instead of using hand-crafted heuristics. In addition, we will explore the use of recurrent architectures to let the agent learn from a history of observations and investigate closed-loop execution of the motion primitives to increase the system's robustness.

References

[1] M. Dogar *et al.*, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.

[2] N. Kitaev *et al.*, "Physics-based trajectory optimization for grasping in cluttered environments," in *IEEE Int. Conf. Robot. Autom.*, 2015, pp. 3102–3109.

[3] W. Bejjani *et al.*, "Occlusion-aware search for object retrieval in clutter," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 4678–4685.

[4] H. Huang *et al.*, "Mechanical search on shelves using a novel "bluction" tool," in *IEEE Int. Conf. Robot. Autom.*, 2022, pp. 6158–6164.

[5] G. J. Pollayil *et al.*, "Planning robotic manipulation with tight environment constraints," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 9385–9392.

[6] M. Stilman *et al.*, "Planning among movable obstacles with artificial constraints," *Int. J. Robot. Res.*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[7] K. Ren *et al.*, "Rearrangement-based manipulation via kinodynamic planning and dynamic planning horizons," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 1145–1152.

[8] H. Zhu *et al.*, "Strategy-based robotic item picking from shelves," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 2263–2270.

[9] M. Costanzo *et al.*, "Can robots refill a supermarket shelf?: Motion planning and grasp control," *IEEE Robot. Autom. Mag.*, vol. 28, no. 2, pp. 61–73, 2021.

[10] S. H. Cheong *et al.*, "Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation," in *IEEE Int. Conf. Robot. Autom.*, 2020, pp. 7791–7797.

[11] C. Nam *et al.*, "Fast and resilient manipulation planning for object retrieval in cluttered and confined environments," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1539–1552, 2021.

[12] H. Huang *et al.*, "Mechanical search on shelves using lateral access x-ray," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 2045–2052.

[13] E. R. Vieira *et al.*, "Persistent homology for effective non-prehensile manipulation," in *IEEE Int. Conf. Robot. Autom.*, 2022, pp. 1918–1924.

[14] D. Batra *et al.*, "Rearrangement: A challenge for embodied ai," *arXiv preprint arXiv:2011.01975*, 2020.

[15] E. Huang *et al.*, "Large-scale multi-object rearrangement," in *IEEE Int. Conf. Robot. Autom.*, 2019, pp. 211–218.

[16] H. Song *et al.*, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 9433–9440.

[17] Z. Pan *et al.*, "Decision making in joint push-grasp action space for large-scale object sorting," in *IEEE Int. Conf. Robot. Autom.*, 2021, pp. 6199–6205.

[18] A. H. Qureshi *et al.*, "Nerp: Neural rearrangement planning for unknown objects," in *Robotics: Science and Systems*, 2021.

[19] A. Krontiris *et al.*, "Dealing with difficult instances of object rearrangement." in *Robotics: Science and Systems*, vol. 1123, 2015.

[20] R. Wang *et al.*, "Uniform object rearrangement: From complete monotone primitives to efficient non-monotone informed search," in *IEEE Int. Conf. Robot. Autom.*, 2021, pp. 6621–6627.

[21] ——, "Efficient and high-quality prehensile rearrangement in cluttered and confined spaces," in *IEEE Int. Conf. Robot. Autom.*, 2022, pp. 1968–1975.

[22] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[23] W. Bejjani *et al.*, "Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 6562–6569.

[24] M. Danielczuk *et al.*, "Segmenting unknown 3d objects from real depth images using mask r-cnn trained on synthetic data," in *IEEE Int. Conf. Robot. Autom.*, 2019.

[25] C. Xie *et al.*, "Unseen object instance segmentation for robotic environments," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1343–1359, 2021.

[26] S. Back *et al.*, "Unseen object amodal instance segmentation via hierarchical occlusion modeling," in *IEEE Int. Conf. Robot. Autom.*, 2022, pp. 5085–5092.

[27] A. Zeng *et al.*, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 4238–4245.

[28] ——, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1307–1319, 2020.

[29] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Adv. Neural. Inf. Process. Syst.* Curran Associates, Inc., 2019, pp. 8024–8035.

[30] Y. Bengio *et al.*, "Curriculum learning," in *Int. Conf. Machine Learning*, 2009, pp. 41–48.

[31] M. Andrychowicz *et al.*, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.