

Hierarchical Policy Learning for Mechanical Search

Oussama Zenkri¹, Ngo Anh Vien², Gerhard Neumann¹

Abstract—Retrieving objects from clutters is a complex task, which requires multiple interactions with the environment until the target object can be extracted. These interactions involve executing action primitives like grasping or pushing as well as setting priorities for the objects to manipulate and the actions to execute. Mechanical Search (MS) [1] is a framework for object retrieval, which uses a heuristic algorithm for pushing and rule-based algorithms for high-level planning. While rule-based policies profit from human intuition in how they work, they usually perform sub-optimally in many cases. Deep reinforcement learning (RL) has shown great performance in complex tasks such as taking decisions through evaluating pixels, which makes it suitable for training policies in the context of object-retrieval. In this work, we first formulate the MS problem in a principled formulation as a hierarchical POMDP. Based on this formulation, we propose a hierarchical policy learning approach for the MS problem. For demonstration, we present two main parameterized sub-policies: a push policy and an action selection policy. When integrated into the hierarchical POMDP’s policy, our proposed sub-policies increase the success rate of retrieving the target object from less than 32% to nearly 80%, while reducing the computation time for push actions from multiple seconds to less than 10 milliseconds.

Video: youtu.be/cioawhgFiLU

Index Terms—Mechanical search, action planner, push policy, hierarchical policy.

I. INTRODUCTION

In recent years, the use of robots in various fields has seen an unprecedented increase. The use cases of robots, however, remained almost unchanged. This is due to the limitation of robots being designed to mainly operate in structured environments. Instead of robots adapting to their environments, work environments are usually adapted to robots. These inconveniences increase their procurement costs and limit their fields of application. In an unstructured environment, such as warehouses or homes, simple actions like grasping objects present a very demanding task for robots, which fails in many cases [2]. Although robots are much faster, much more precise, and much stronger than humans, they are significantly outperformed by humans when it comes to manipulation tasks. Humans show this superior performance thanks to the disposal of a nearly endless repertoire of motion primitives that they can easily perform, as well as the ability to safely predict the outcome of any motion primitive [3], which robots lack.

In this paper, we want to learn how to grasp a specific object from a heap of cluttered objects. The target object might not

be visible in the beginning and other objects might need to be removed or pushed away to make the target object visible and reachable. We will base our work on an existing framework from a recent work called Mechanical Search [1], where the authors combine different algorithms for object recognition and learning grasp points. For object recognition, SD Mask-RCNN [4] and a Siamese network [5] are used, while Dex-Net [6] is used for selecting the grasp point. In this approach, the objects to be removed are currently selected via heuristics.

In this work, we first formulate the MS problem in a principled formulation as a hierarchical POMDP. Based on this formulation, we propose a hierarchical policy learning approach. Policies higher in the hierarchy, e.g. action selection policies, will be optimized through the use of hierarchical dynamic programming which carries Bellman updates based on the value functions of policies lower in the hierarchy (called sub-policies), e.g. push and grasp policies. Policies at each layer in the hierarchy can be optimized in bottom-up fashion [7] or jointly [8]. For demonstration, we present two main parameterized sub-policies: a push policy and an action selection policy. We propose to use reinforcement learning to optimize these policies. When integrated into the global hierarchical policy to do MS, our proposed solution increases the success rate of retrieving the target object in a clutter of 20 objects from less than 32% to nearly 80%.

II. RELATED WORK

Many research projects, which deal with developing the cognitive abilities of robots have made significant improvements in the fields of perception, planning, and control. However, less progress has been made on more complex tasks, which require a combination of these fields, like retrieving objects in unstructured environments, such as warehouses or offices.

An early work by Li et al. [9] demonstrates how modelling the problem of object search in clutter as a POMDP improves action planning and thus reduces the number of required steps to retrieve a target object. However, this work mainly deals with the problem of partial occlusion.

Danielczuk et al. [1] propose mechanical search that can deal with extracting objects from clutter. Mechanical search uses SD Mask R-CNN [4], a depth-based category-agnostic segmentation algorithm, to distinguish objects and a Siamese network [5] to identify the target object. It uses Dex-Net [6], [10] to plan grasps and a heuristic policy to plan pushes [11]. All action policies produce a quality value for the action they plan. Which action is to apply on which object is also decided heuristically. The target object, if found, has always the highest

¹ Karlsruhe Institute of Technology (KIT)

² Bosch Center for Artificial Intelligence (BCAI)

priority for manipulation. If not found or impossible to apply actions on the target object, the other detected objects are sorted in descendent order according to their visible area. In a second work, Danielczuk et al. [12] propose a follow-up idea to improve the search by estimating the occupancy distribution of the target object, which is often occluded from the scene. A further follow-up work from the same authors tries to train a push policy [13] via learning from demonstration and RL, which shares a similar direction to our proposed approach. However, we resort to motion primitives with task parameterization to achieve sample efficiency instead of using teacher guidance that is not always available.

Sarantopoulos et al. [14] introduce a framework for singulating objects through pushing. It incorporates two push primitives, which evaluate a latent representation of the visual state to generate push candidates for the target object and the obstacles. The high-level policy decides which push to apply. While singulation can speed up object retrieval, this approach is only beneficial in scenes with enough free space. Another related work by Dogar et al. [3] which also deals with extracting objects from clutter presents a very similar approach. It uses a repertoire of human-inspired primitives like pushing, grasping, end effector motions, and combinations. Novkovic et al. [15] introduces a framework for uncovering objects in cluttered scenes through interactive perception. The used policy processes an encoded volumetric truncated signed distance field (TSDF) representation of the RGB-D observation to generate some end effector displacement. Pan et al. [16] introduce a bi-level motion planner, which deals with making optimal decisions in a joint push-grasp action space for object sorting. Zeng et al. [17] present a new method for planning pushes and grasps using two end-to-end deep networks to capture complementary pushing and grasping policies that benefit from each other via reinforcement learning. Another work by Deng et al. [18] propose a deep RL approach for pushing and picking in clutter. The RGB-D observation is transformed into an affordance map, which is used to determine if a grasp is possible and where to perform it. The agent disturbs the scene by performing a push action if grasping fails or if no grasping is possible. Berscheid et al. [19] and Feldman et al. [20] present a vision-based algorithm for learning the most rewarding pose for applying object manipulation primitives. These algorithms are targeted to learn a policy that chooses between shifting and grasping actions in a bin-picking setting. However these methods are not designed for searching for a target object.

III. PROBLEM FORMULATION

Different from the original formulation from Danielczuk et al. [1], we formulate the object search problem as a hierarchical POMDP [7], [21]. Similar to a standard POMDP as defined by Danielczuk et al. [1], a hierarchical POMDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{R}, \mathcal{Y})$.

- **State** $s \in \mathcal{S}$: a set of ground-truth information of the environment, i.e. all object’s geometry and pose, robot’s states, all sensor’s states.

- **Action** $a \in \mathcal{A}$: a set of robot commands, e.g. a next end effector (EE) pose.
- **Observation** $y \in \mathcal{Y}$: a set of sensor data, which is an RGB-D image in our setting.
- **Transition** \mathcal{T} : defines a state transition given action a , i.e. $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ where $\mathcal{T}(s', s, a) = p(s'|s, a)$. s' denotes the following state of s .
- **Observation function** \mathcal{O} : defines an observation distribution given ground-truth state s , i.e. $\mathcal{O}(y, s) = p(y|s)$.
- **Reward function** \mathcal{R} : defines a reward given state s and action a , i.e. $R(s, a)$.

Finding a global policy for the above POMDP problem is hard [7]. In literature, there has been much work attempting to reduce problem complexity by proposing hierarchical approaches or action decomposition.

In principle, a hierarchical POMDP can be decomposed as a hierarchy of partially observable semi-MDPs (POSMDPs) [22]. In particular, assuming a hierarchical POMDP can be solved with a hierarchical policy $\pi = \{\pi_0, \pi_1, \dots, \pi_k\}$, where $\pi_0, \pi_1, \dots, \pi_k$ are called sub-policy, macro action, parameterized motion primitives, etc. in different contexts. Each sub-policy is targeted to solve a sub-POMDP problem (sub-task) which has much lower complexity than the global task. This principled policy decomposition can result in hierarchical Bellman updates, hence hint a principled way to optimize a hierarchically or recursively optimal policy (the type of optimality would depend on how each sub-task’s reward function is defined), i.e. via hierarchical dynamic programming.

In regard to the approach of Danielczuk et al. [1], the authors have also come up with a hierarchical policy design. The global policy π is decomposed into a search policy π_0 whose only child action is the object selection policy π_1 . Policy π_1 ’s child actions are segmented objects $\{\pi_2, \dots, \pi_{N+2}\}$ (assuming there are N objects in the scene at a particular time t). Each policy π_i ($2 \leq i \leq N + 2$) has the same action policy that contains a set of two actions: grasp action policy π_{N+3} and push action policy π_{N+4} . Each of the grasp or push action policies are parameterized motion primitives that would use low-level action $a \in \mathcal{A}$ as primitive actions. However, Danielczuk et al. [1] have not yet discussed if each sub-policy π_i is derived from solving a well-defined sub-POMDP problem. Among them, grasp action policy π_{N+3} is a pretrained Dex-Net policy network [6] which was shown to come from a POMDP grasping problem. The other sub-policies are heuristic and hard-coded, therefore not designed as a solution to a well-defined sub-POMDP problem. Thus, this approach could have a great practical benefit but might lack the flexibility of extension to more powerful solutions, e.g. using hierarchical RL.

Based on our above hierarchical POMDP formulation and action decomposition, we propose two extensions that transform i) the action selection problem to a POMDP problem with a trainable *action selection policy* (depending on its child actions), and ii) the push action to a POMDP problem with a trainable *push policy*. Our following proposals assume separate

training for simplicity, however, it can also be trained in an end-to-end hierarchical POMDP RL fashion [8].

IV. PUSH POLICY

Danielczuk et al. [1] use the free space policy (FSP) [11], as a push primitive in the MS pipeline. The FSP aims to push the target object (TO) to the freest space in the bin. This would increase the free space around the TO and consequently increases the probability of a successful grasp. Danielczuk et al. conjecture that this policy performs suboptimal and suggest replacing it with more effective push primitives, which can learn from simulation. Our observations validate this assumption. While the FSP performs well in small heaps, it shows poor performance in cluttered scenes where increasing the free space around an object is difficult. Moreover, the FSP fails at inferring a push action for many observations and takes multiple seconds to compute one push action. These drawbacks motivate us to develop a new push policy, which learns pushing from simulation. First, we introduce a problem statement for the linear pushing, then we introduce our method for solving the push problem.

A. Problem Statement

We formulate the general linear push problem as a sub-POMDP defined by $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Y})$, where it shares the state space \mathcal{S} of the main problem defined in Section III. For this problem we precisely define the observations, the action set, and the reward function. All other aspects of the problem formulation are sufficiently captured by the main POMDP.

- **Observation** $y \in \mathcal{Y}$: a downscaled crop from the depth image centered around the target object. The crop is sized 220×220 (px) and downscaled to 40×40 (px).
- **Action** $a \in \mathcal{A}$: a six-dimensional continuous action defined by $x_{rel}, y_{rel}, \sin \alpha_{Push}, \cos \alpha_{Push}, \sin \phi, \cos \phi \in [-1, 1]$. The action encodes the relative position of the EE at push start (x_{rel}, y_{rel}) , the push direction α_{Push} and the yaw angle of the EE ϕ . Values x_{rel} and y_{rel} are relative to the observation image. The absolute coordinates in the depth image (u, v) can be deduced from the size and coordinates of the crop. The first four variables of the action space are visualized in Fig. 1. The push start position \mathbf{p}_{Start} is obtained through deprojecting u, v and the estimated depth z . The push distance is 10 cm. The push end point \mathbf{p}_{End} is inferred from \mathbf{p}_{Start} and α_{Push} .
- **Reward** $r \in \mathcal{R}$: a value quantifying the change in free space around the OOI. The reward function is defined as the weighted sum of changes in free space around each object present in the bin:

$$r_t = \frac{10 \cdot \Delta_{FS, O_I}(t) + \frac{1}{N-1} \cdot \sum_{i=1, i \neq I}^N \Delta_{FS, O_i}(t)}{11} \quad (1)$$

where, N denotes the number of objects in the bin. O_I denotes the OOI. Δ_{FS, O_i} denotes the change in free space (FS) around object O_i after executing the push action as

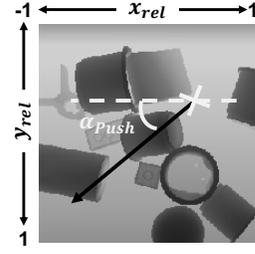


Fig. 1: Visualization of the action space variables for the push policy. The position of EE at push start (x_{rel}, y_{rel}) is shown as a white cross. The push direction is defined by α_{Push} . The EE-yaw angle ϕ is not represented in this figure.

denoted in Eq. 2. We reward the increase of free space around all objects, while focusing on the OOI.

$$\Delta_{FS, O_i}(t) = FS_{O_i}(t) - FS_{O_i}(t-1). \quad (2)$$

Here, $FS_{O_i}(t)$ denotes the free space around O_i at time t and is computed as the sum of the masked distance transform of the bin free space for O_i (Fig. 2d), normalized by the area of O_i

$$FS_{O_i}(t) = \frac{\sum (\mathbf{M}_{O_i}(t) \odot \mathbf{DT}_{BFS, O_i}(t))}{\sum \mathbf{M}_{O_i}(t)} \quad (3)$$

where, \mathbf{DT}_{BFS, O_i} denotes the result of the distance transform of the bin free space (BFS) mask for object O_i (Fig. 2c). $\sum \mathbf{M}_{O_i}$ is a scalar denoting the area of the binary mask \mathbf{M}_{O_i} of object O_i (Fig. 2a). The operator \odot denotes the element-wise multiplication. The BFS-mask (Fig. 2b) results from the subtraction of all observed object masks from the bin bottom (BB) mask except for the OOI:

$$\mathbf{M}_{BFS, O_i}(t) = \mathbf{M}_{BB} - \sum_{i=1, i \neq I}^N \mathbf{M}_{O_i}(t) \quad (4)$$

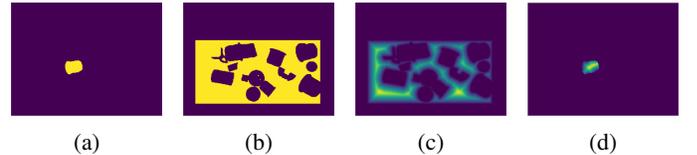


Fig. 2: Elements of the computation of the free space for object O_i . (a) Binary mask of object O_i . (b) Bin free space mask. (c) Distance transform of \mathbf{M}_{BFS, O_i} . (d) Masked \mathbf{DT}_{BFS, O_i} .

B. Push Policy Learning

For our push policy we use an actor-critic architecture, which consists of two branches: an actor-network and a critic-network. The actor takes a state observation as input and returns an action according to a policy π_θ . The critic serves as an action-value function and returns a quality value for the computed action given the observed state. These two networks are optimized jointly during the training. Our architecture also includes an encoder, which transforms the

input image of size 40×40 (px) to a feature vector of size 98. This feature vector serves as the state observation for the actor and the critic. We train the encoder jointly with the other networks. The architecture of our push policy is shown in Fig. 3. The generated action vector $\mathbf{a}_t = (x_{rel,t}, y_{rel,t}, \sin \alpha_{Push,t}, \cos \alpha_{Push,t}, \sin \phi_t, \cos \phi_t)$ is concatenated with the state observation and forwarded to the critic-network. We optimize our policy using the soft actor-critic (SAC) algorithm [23].

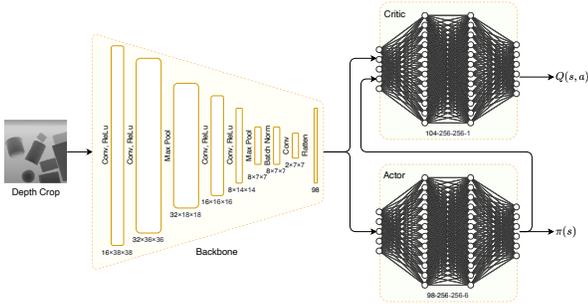


Fig. 3: Architecture of our push policy. The network consists of a CNN encoder for feature extraction, a fully connected actor-network serving as policy, and a fully connected critic-network, which serves as an action-value function.

V. ACTION SELECTION POLICY

The heuristic action selection policy (ASP), introduced in [1] has a simple tree structure, which prioritizes grasping over pushing. If neither actions are possible, then the current object is skipped. The policy uses static threshold values $q_{Grasp,Thresh}$ and $q_{Push,Thresh}$ to determine the priorities of the actions, which can result in suboptimal decisions, especially in cases of wrong quality estimations. This suboptimal behaviour can be observed in the physical rollouts introduced in [1], where the heuristic policy is outperformed by humans, who require less than three actions to succeed in more than 50% of the trials, while the introduced heuristic policy requires at least five actions to reach the same success rate. The superior human performance results i.a. from making better action selection decisions as shown in Table I. The table shows a remarkable difference in the proportions of executed actions. This motivates us to develop a new ASP, which learns a better action selection strategy from trial and error. First, we formulate a problem statement for the action selection, then we introduce our method for solving this problem.

TABLE I: Proportions of taken actions by the heuristic ASP and humans.

	Heuristic ASP*	Human ASP*
Parallel-Jaw Grasp	18.4%	15.9%
Suction Grasp	76%	58.2%
Push	5.6%	25.9%

*: the values result from 50 physical rollouts. Reported in [1]

A. Problem Statement

The objective of our ASP is to select one of the available action primitives such that the number of required actions to extract a TO is minimized. We formulate the problem of action selection as a sub-POMDP problem that also shares the same state space as the main POMDP problem defined in Section III. For this sub-POMDP task (action selection), we specify the observations, the action set and the reward function.

- **Observations:** the environment returns the following observation set:

- 1) **Depth Image:** a 40×40 (px) rescaled crop of the depth image centered around the OOI. The crop size is 220×220 (px), which captures sufficient context from the original image of size 480×640 (px).
- 2) **OOI-Mask:** a 40×40 (px) rescaled crop of the OOI-mask centered around the OOI. The crop size is 220×220 (px).
- 3) **Relative Coordinates:** $x_{rel}, y_{rel} \in [-1, 1]$ designate the relative coordinates of the TO in the original depth image. The absolute position of the TO is inferred from these values.
- 4) **Action Qualities:** $q_{Grasp}, q_{Push} \in \{-1, [0, 1]\}$ the quality metrics returned by the action policies. While the policies return values in $[0, 1]$, we set the quality value to -1 , if the policy could not compute an action for the OOI. This signals the agent that the corresponding action is infeasible. This behavior is reinforced with the used reward function (Eq. 5). Selected infeasible actions are changed to *Skip*.

- **Actions:** the agent selects one of the following actions:
 - 1) **Skip:** the robot does not apply any action on the current object. Another object will be evaluated.
 - 2) **Grasp:** the robot grasps the current object. If the OOI is marked as TO, the object is dropped in a dedicated spot. Otherwise, the robot drops the object in a secondary bin. We assume the grasp policy is a pretrained Dex-Net 2.0 policy network [6].
 - 3) **Push:** the robot pushes the OOI according to the plan returned by the push policy. This push action is an optimized push policy discussed in Section IV.
- **Reward:** we define the reward function as

$$r_t = \begin{cases} 20 & , \text{if the TO is successfully extracted,} \\ -10 & , \text{if an infeasible is action selected,} \\ -1 & , \text{Otherwise.} \end{cases} \quad (5)$$

B. Method

The objective of the ASP is to select the motion primitive (grasp vs. push) to be executed, if applicable, which minimizes the number of required actions to retrieve a TO. This task has a finite discrete action space, which makes a Q-Learning algorithm a simple and effective RL method to deal with it. Thus, we use a deep Q-network (DQN) [24] architecture for our policy. Due to the property of the observation, which

consists of a mixture of images and scalars, our network has two input branches. The first one includes an encoder for extracting features from the input images. The second branch feeds the input scalars directly in the Q -network. The network outputs an action-value for each of the possible actions $\{Skip, Grasp, Push\}$. The action with the highest Q -value is selected for execution. The architecture of our action selection agent is shown in Fig. 4. The used encoder shares the same architecture as the one in our push policy (sec. IV-B).

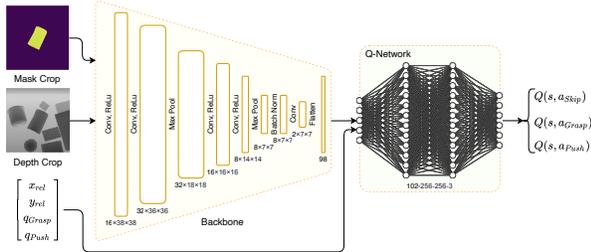


Fig. 4: Architecture of our ASP. The encoder extracts features from the depth image and the OOI-mask and outputs a feature vector of size 98. The feature vector is concatenated with the scalar input data and forwarded to the Q -network, which outputs a quality value for each state-action pair.

VI. EXPERIMENTS

A. Environment

In our work, we use a simulated environment running on the PyBullet physics engine [25] for training our agents and benchmarking the different policies. The scene, shown in Fig. 5, consists of a simulated Franka Emika Panda robot equipped with a parallel-jaw gripper. The robot is placed on top of a table close to a bin, where the objects to manipulate are placed. A secondary bin, for dropping secondary objects, is placed beside the main bin. The used object models are a subset of the YCB dataset [26]. The scene is perceived through a simulated Intel RealSense D-435 RGB-D camera [27].

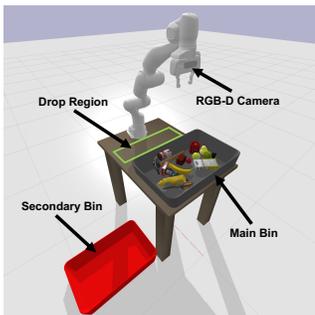


Fig. 5: Overview of the work environment. The designated drop region for the TO is marked in green.

B. Experiment Description

To evaluate our policies, we compare their performances to the ones of the heuristic policies, which were introduced in the original MS pipeline [1]. The trials are initialized as follows:

- 1) Randomly, select a TO from the list of available objects.
- 2) Randomly, select other objects from the remaining objects, until the benchmark heap size is reached.
- 3) Place the selected objects, collision free, at different locations and heights above the bin.
- 4) Simultaneously, drop all objects in the bin.
- 5) Allow 5 seconds for the objects to settle before making the first observation.

After initializing the scene, the agent is allowed to sequentially apply pushes and parallel-jaw grasps to retrieve the TO. The agent can push objects in the bin or extract objects, different from the TO, and place them in the secondary bin. The trial ends when one of the stopping criteria is met:

- The TO is dropped at the designated drop location.
- The TO is neither in the bin nor in the drop location.
- The number of executed actions exceeded 25.

C. Training

1) *Push Policy*: We train our push policy on clutters of ten objects, which we place randomly at different locations in the bin. We set the length of an episode to five push trials to speed up training. The agent is trained for 12k episodes, i.e. a total of 60k steps. The policy is updated five times after each step. The policy converges after 10k episodes.

2) *Action Selection Policy*: We train our ASP on heaps of size 20. An episode is automatically terminated after 25 steps. Our agent is trained for 300 episodes. The agent is updated 20 times after each step. The policy converges after 100 episodes.

D. Evaluated Setups

We evaluate four different setups of the MS pipeline, which contain the four possible combinations illustrated in Table II. All setups retain the same components of the MS pipeline, except for the push policy, and the ASP. Each setup is evaluated on two different heap sizes: 10 and 20 objects.

TABLE II: Policy combinations of the evaluated setups

	Heu. ASP	Our ASP	Heu. Push Pol. ⁺	Our Push Pol.
Setup 1	×		×	
Setup 2	×			×
Setup 3		×	×	
Setup 4		×		×

+ : free space policy (FSP).

VII. RESULTS

We conducted 100 simulated experiments for each policy combination and heap size resulting in 800 rollouts. The four different policy combinations are compared against each other according to Section VI. The results of the experiments are visualized in Fig. 6 and also illustrated in Table III.

Fig. 6a shows the average number of actions required by each policy combination to retrieve the TO for heaps of size 10 and 20. The four combinations perform similarly on the small heaps. In the larger heaps, Setup 1 performs the worst with a mean of 10.06 actions. Setup 3 requires only 4.92 actions.

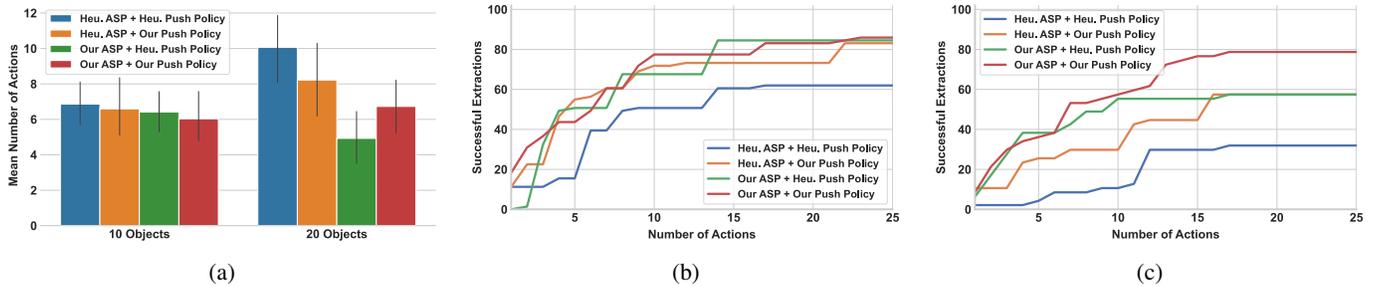


Fig. 6: Performance of our policies compared to the heuristic policies of the original MS pipeline. (a) Average number of required actions for object retrieval for heaps of 10, and 20 objects. Successful extractions as function of the number of executed actions for simulated heaps of (b) size 10, and (c) size 20. Substituting the FSP and the ASP with our policies results in similar or better performance than swapping only one of them. This effect is more pronounced for larger heaps.

The success rates of the policy combinations as a function of the number of executed actions for heaps of size 10 and 20 are shown in Fig. 6b and Fig. 6c respectively. Both figures show that substituting the FSP by our RL-based push policy (Setup 2) leads to higher success rates and fewer required actions to retrieve the TO. Fig. 6b shows that, when our policies are used (Setups 2, 3, and 4), the success rates exceed 40% within five actions, while the success rate of the original MS pipeline (Setup 1) remains below 20% for the same number of actions. If our ASP is used (Setups 3 and 4), the success rate reaches 80% within 17 actions, while the original MS pipeline settles at 60%. For heaps of size 20, the success rate of the original MS pipeline (Setup 1) settles below 32%. If the FSP and ASP are swapped with our policies (Setup 4), the success rate reaches 79% within 17 actions.

TABLE III: Means and standard deviations of required actions for a successful object retrieval in heaps of size 10 and 20

	Mean ₁₀	Std. Dev. ₁₀	Mean ₂₀	Std. Dev. ₂₀
Setup 1	6.86	4.22	10.06	3.84
Setup 2	6.59	6.24	8.22	5.46
Setup 3	6.42	4.22	4.92	3.71
Setup 4	6.03	5.27	6.73	4.72

The comparable performance of all setups for heaps of size 10 is due to the small size of the heap, which makes most of the objects well exposed. Consequently, the TO can be found easily. Setup 4 reaches higher success rates at higher numbers of actions. This biases the mean number of required actions towards higher values, thus making it appear worst than Setup 3. However figure 6c shows that Setup 4 outperforms all other setups at the majority of the given number of actions.

TABLE IV: Proportions of executed actions for different ASPs

	Original ASP	Human ASP*	Our ASP
Parallel-Jaw Grasp	86.3%	15.9%	67.9%
Suction Grasp	-	58.2%	-
Push	13.7%	25.9%	32.1%

*: the values result from 50 physical rollouts. Reported in [1].

Table IV compares the proportions of executed actions by

the original ASP, Humans, and our ASP. It shows that the proportion of executed pushes by the heuristic ASP is much lower than the proportion executed by humans as pointed out by Danielczuk et al. [1]. While humans make more use of pushing through selecting it more than 25% of the time, less than 14% of the selected actions by the heuristic ASP are pushes. Our ASP, however, shows more human-like action choice through deciding for pushing in 32% of the time.

VIII. CONCLUSION AND FUTURE WORK

We propose a hierarchical POMDP formulation for the mechanical search problem where each sub-policy can be trainable and integrated to be optimized in a principled way. For demonstration, we introduce a new push policy, which we optimize using soft actor-critic. The aim of our push policy is to maximize the free space around a target object. We also introduce a new action selection policy, which makes use of the deep Q-network architecture to select the best action primitive to be applied for a given observation. Combined, our two policies show a significant decrease in the number of required actions to extract an object, compared to the heuristic policies introduced in [1]. This effect is more pronounced for large heaps. The success rate of object retrieval for a frame of 17 actions increases from less than 32% using the original MS pipeline to nearly 80% when using our policies. Substituting the heuristic free space policy introduced in [11] by our push policy substantially reduces the inference time from multiple seconds to less than 10 milliseconds. Our action selection policy takes significantly more use of the push primitive than the heuristic policy in [1] and shows similar action proportions to the ones selected by humans.

The confirmation of these results on real-world experiments, the introduction of a new learning object selection policy for ranking the objects to manipulate, and the evaluation of a joint end-to-end training of all hierarchical policies are subject of future work.

IX. ACKNOWLEDGMENTS

This research has been supported by HEAP (Human-Guided Learning and Benchmarking of Robotic Heap Sorting, CHIST-ERA, grant no: EP/S033718/2).

REFERENCES

- [1] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Multi-step retrieval of a target object occluded by clutter," in *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*. IEEE, 2019, pp. 1614–1621.
- [2] D. Katz, J. Kenney, and O. Brock, "How can robots succeed in unstructured environments," in *In Workshop on Robot Manipulation: Intelligence in Human Environments at Robotics: Science and Systems*. Citeseer, 2008.
- [3] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.
- [4] M. Danielczuk, M. Matl, S. Gupta, A. Li, A. Lee, J. Mahler, and K. Goldberg, "Segmenting unknown 3d objects from real depth images using mask r-cnn trained on synthetic data," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7283–7290.
- [5] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," 2015.
- [6] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *arXiv preprint arXiv:1703.09312*, 2017.
- [7] J. Pineau, "Tractable planning under uncertainty: Exploiting structure," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, 2004.
- [8] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, pp. 3675–3683, 2016.
- [9] J. K. Li, D. Hsu, and W. S. Lee, "Act to see and see to act: Pomdp planning for objects search in clutter," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 5701–5707.
- [10] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning," in *2018 IEEE International Conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 5620–5627.
- [11] M. Danielczuk, J. Mahler, C. Correa, and K. Goldberg, "Linear push policies to increase grasp access for robot bin picking," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2018, pp. 1249–1256.
- [12] M. Danielczuk, A. Angelova, V. Vanhoucke, and K. Goldberg, "X-ray: Mechanical search for an occluded object by minimizing support of learned occupancy distributions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 9577–9584.
- [13] A. Kurenkov, J. Taglic, R. Kulkarni, M. Dominguez-Kuhne, A. Garg, R. Martín-Martín, and S. Savarese, "Visuomotor mechanical search: Learning to retrieve target objects in clutter," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 8408–8414.
- [14] L. Berscheid, P. Meißner, and T. Kröger, "Robot learning of shifting objects for grasping in cluttered environments," in *2019 IEEE/RSJ*
- [14] I. Sarantopoulos, M. Kiatos, Z. Doulgeri, and S. Malassiotis, "Total singulation with modular reinforcement learning," *IEEE Robotics and Automation Letters*, pp. 1–1, 2021.
- [15] T. Novkovic, R. Pautrat, F. Furrer, M. Breyer, R. Siegart, and J. Nieto, "Object finding in cluttered scenes using interactive perception," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8338–8344.
- [16] Z. Pan and K. Hauser, "Decision making in joint push-grasp action space for large-scale object sorting," *arXiv preprint arXiv:2010.10064*, 2020.
- [17] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.
- [18] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, "Deep reinforcement learning for robotic pushing and picking in cluttered environment," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 619–626.
- [19] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, "Deep reinforcement learning for robotic pushing and picking in cluttered environment," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 612–618.
- [20] Z. Feldman, H. Ziesche, N. A. Vien, and D. D. Castro, "A hybrid approach for learning to shift and grasp with elaborate motion primitives," *CoRR*, vol. abs/2111.01510, 2021. [Online]. Available: <https://arxiv.org/abs/2111.01510>
- [21] N. A. Vien and M. Toussaint, "Hierarchical monte-carlo planning," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, B. Bonet and S. Koenig, Eds. AAAI Press, 2015, pp. 3613–3619.
- [22] C. C. White, "Procedures for the solution of a finite-horizon, partially observed, semi-markov optimization problem," *Operations Research*, vol. 24, no. 2, pp. 348–358, 1976.
- [23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1856–1865.
- [24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation in robotics, games and machine learning," 2017.
- [26] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.
- [27] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 1–10.