
Article:
Bejjani, W orcid.org/0000-0002-6129-2460, Leonetti, M orcid.org/0000-0002-3831-2400 and Dogar, MR orcid.org/0000-0002-6896-5461 (2021) Learning image-based Receding Horizon Planning for manipulation in clutter. Robotics and Autonomous Systems, 138. 103730. ISSN 0921-8890

https://doi.org/10.1016/j.robot.2021.103730

Learning Image-Based Receding Horizon Planning for Manipulation in Clutter

Wissam Bejjani[a,*], Matteo Leonetti[a] and Mehmet R. Dogar[a]

[a]School of Computing, University of Leeds, LS2 9JT, Leeds, UK

ARTICLE INFO

Keywords:
Manipulation in clutter
Physics-based manipulation
Heuristic learning
Receding horizon planning
Imitation and Reinforcement learning
Abstract state representation

ABSTRACT

The manipulation of an object into a desired location in a cluttered and restricted environment requires reasoning over the long-term consequences of an action while reacting locally to the multiple physics-based interactions. We present Visual Receding Horizon Planning (VisualRHP) in a framework which interleaves real-world execution with look-ahead planning to efficiently solve a short-horizon approximation to a multi-step sequential decision making problem. VisualRHP is guided by a learned heuristic that acts on an abstract colour-labelled image-based representation of the state. With this representation, the robot can generalize its behaviours to different environment setups, that is, different number and shape of objects, while also having transferable manipulation skills that can be applied to a multitude of real-world objects. We train the heuristic with imitation and reinforcement learning in discrete and continuous actions spaces. We detail our heuristic learning process for environments with sparse rewards, and non-linear, non-continuous, dynamics. In particular, we introduce necessary changes for improving the stability of existing reinforcement learning algorithms that use neural networks with shared parameters. In a series of simulation and real-world experiments (video available on https://youtu.be/raKHTnJLikQ), we show the robot performing prehensile and non-prehensile actions in synergy to successfully manipulate a variety of real-world objects in real-time.

1. Introduction

Many robotics applications require robots to act in unstructured and cluttered environments with uncertain dynamics. Examples include reaching for specific objects on a warehouse shelf, sorting waste, and grasping a food item from a fridge [12, 14, 44]. In most of these applications, robots must react, and even leverage, multi-object physical interactions in real-time. For instance, in Fig. 1 the robot is tasked with manipulating in a planar space the orange fruit to the target region. Accomplishing this task involves interpreting the task goal, representing the environment, reasoning over the environment's physics, and executing in real-time a long sequence of prehensile and non-prehensile actions while satisfying constraints such as not to drop any of the objects off the surface edges.

The multi-step, sequential, and high dimensional nature of such tasks makes sampling-based planning an attractive option for solving the problem. Nonetheless, the uncertainty in modelling the physics (both in object-to-object and object-to-surface friction) in a cluttered environment would necessitate real-time closed-loop decision making [35]. Replanning makes the computation cost of sampling-based planning impractical for real-time applications as they treat every new planning instance independently of previous experiences and they also have to solve the problem all the way to the goal. Instead, shortening the planning horizon can reduce the computation cost and mitigate the uncertainty associated with increased clutter density.

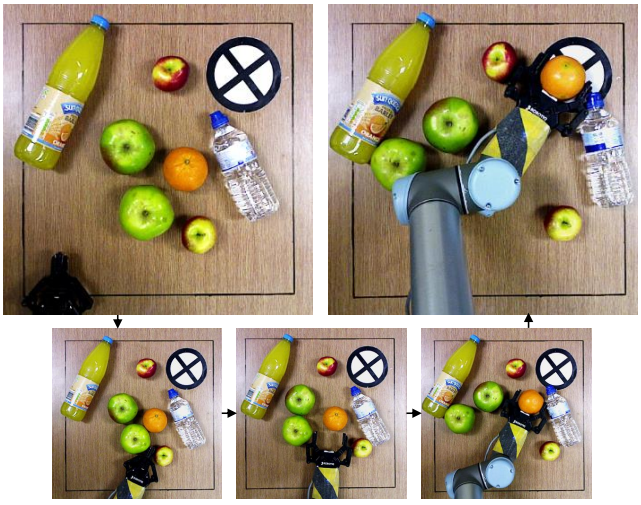


Fig. 1. The robot is tasked with moving, whithin the surface edges, the orange fruit to the target region using prehensile and non-prehensile planar actions.

Planning with a short horizon requires an efficient local search, and a cost-to-go function expressing the expected cost for the rest of the plan beyond the horizon. Receding Horizon Planning (RHP) is a planning framework aimed at continual planning with a limited look-ahead [38]. Heuristic-based RHP offers desirable properties for achieving physics-aware manipulation skills in closed-loop. It relies on a heuristic to perform a local search using the forward model of a physics simulator, and to evaluate the potential consequences of finite sequences of actions, before executing the first action of a chosen sequence [26]. We adopt this paradigm,

*This work has received funding from the UK Engineering and Physical Sciences Research Council under grant EP/R031193/1.

*Corresponding author

wissam.bejjani@hotmail.com (W. Bejjani); m.leonetti@leeds.ac.uk (M. Leonetti); m.r.dogar@leeds.ac.uk (M.R. Dogar)

and propose using a heuristic, learned with Reinforcement Learning (RL), to guide the local search and estimate the cost-to-go.

In image-based RL, RGB data are mapped to control actions. This framework enables greater generalization by making geometric features of the environment available to the learner while also relieving the algorithm designer from having to hard-code task-relevant features that might hinder the robot in leveraging the full dynamics of the environment [23]. It remains challenging, however, for the neural model of an RL system to capture the complexity introduced by non-linear and non-continuous dynamics of the physics environment whilst proposing actions with long-term consequences. Furthermore, models that directly act on real-world images to manipulate a specific object, defined for example by its shape or colour [49], lack transferability to tasks with different desired object attributes.

In this article, we propose **VisualRHP** an approach that combines the advantages of both image-based learning and heuristic-based RHP. We design a framework around VisualRHP to interleave real-world execution with abstract image-based look ahead planning in a physics simulator. The real-world state is abstracted to a colour-labelled image representation rendered from the simulator state, enabling *generalizable* and *transferable* manipulation skills. In the simulator, VisualRHP uses a learned image-based heuristic that acts on the abstract state representation to efficiently solve a short horizon approximation to a multi-step sequential decision making problem. The acquired skills by VisualRHP combine prehensile and non-prehensile manipulation actions that generalize to different number of objects with different shapes, and transfer to tasks with a different desired object to manipulate in the real world.

Our contributions build on our research on heuristic learning for RHP in a *discrete action space* [6] and on learning transferable manipulation skills by means of an abstract state representation [5]. We extend these contributions to learning manipulation actions in a *continuous action space* reducing by a third the number of actions required to solve a manipulation problem compared to a discrete action space. The two heuristic learning approaches, in discrete and in continuous action spaces, presented in this article put forth modifications on existing Imitation Learning (IL) and RL methods to improve on the stability of learning algorithms in sparse rewards environments with non-linear and non-continuous dynamics.

The contributions of this work are (*i*) a framework that integrates of real-world execution with physics-based look-ahead planning in simulation (Fig. 2 and Sec. 3), (*ii*) an abstract image-based representation that uses colour-labelling to represent the state of the manipulation task (Sec. 5.1), (*iii*) and the VisualRHP algorithm that uses an image-based heuristic to run RHP in discrete and continuous action spaces (Sec. 5.2). While our overall framework is agnostic to the particular way the heuristic is learned, (*iv*) we detail a stable heuristic learning process in Sec. 6. These contributions culminate in having robust and transferable manipulation skills to different desired objects while generalizing over different environment settings.

The remainder of this work is structured as follows. Sec.2 presents and locates our work w. r. t. related literature. An overview of the framework is described in Sec.3. Sec.4 presents the formalism adopted in this work. The details of the control loop including VisualRHP is found Sec.5. The IL and RL heuristic learning algorithms are presented in Sec.6. The simulation and real-world experiments are presented in Sec.7 and Sec.9. Concluding comments are found in Sec.10.

2. Related Work

Manipulation in cluttered spaces has long been approached with planning-based techniques. Planners such as [18, 30, 29, 13, 4] adopt an approach of motion planning followed by **open-loop** execution to solve the task. In particular, Haustein et al. [18] adopt sampling-based planning to solve manipulation in clutter problems. They propose reducing the search space of kinodynamic Rapidly exploring Random Trees (RRT) by planning over statically stable environment states, while allowing for physical interaction in-between these states. We use a similar kinodynamic RRT planner to generate demonstrations in different task instances. There are planners which also take uncertainty into account before the generation of the motion trajectory [13, 34, 25, 43], but these planners typically rely on uncertainty reducing actions, which generate a conservative sequence of actions, limiting the robot from using the complete dynamics of the domain. Alternatively, to avoid the uncertainty associated with multiple objects interacting in a cluttered environment, planning approaches have been developed to avoid contact altogether with environment obstacles. Finding a collision free trajectory has been the common theme in many of the approaches presented at the Amazon Picking Challenge [20, 42]. Most recently, Kimmel et al. [28] motivate a two-step planning approach, first in the task space and then in the robot joint configuration space, to find a collision-free trajectory to a stable grasp pose and for retrieving the desired object. Avoiding all obstacles remains, however, impossible (and often undesirable) in many environment setups, and with that comes the necessity of a reactive system.

Learning-based approaches have seen significant progress in recent years in developing closed-loop physics-based non-prehensile manipulation policies. To learn a robust behaviour, Kloss et al. [31] combine an image-based learning approach with an analytical model of a pushing task. They train a neural network, on visual sensory input, to output the appropriate physical parameters to the analytical model that is controlling the robot. Also for a pushing task, Peng et al. [46] propose randomizing the physics parameters in the simulator, such that the learned behaviour avoids exploiting the inaccuracy of the physics model in the simulator. Clavera et al. [11] argue for a modular approach where sensing, policy, and controller are designed separately to ease the skill transfer from simulation to the real world. These approaches have proven capable in real-world manipulation. However, they

are designed for pushing a single object and their extension to multi-object environments is yet to be explored.

The state representation, that is, the features on which decisions are made, plays a major role in shaping the learned behaviour. When deciding on the state representation, the key design choice lies in balancing the trade-off between having a representation that is *expressive* enough to capture the state, while also providing a space that is *efficiently searchable* [10]. Raw sensory representations, such as in the form of realistic images, carry an abundant amount of information, i. e., highly expressive, and does not require feature engineering. Image-based representations can take advantage of the spatial generalization of Convolutional Neural Networks (CNN) to learn implicitly spatial features that allow for greater task generalization [56]. Domain randomization is often used in conjunction with realistic image-based representations to guide the learning process to capture features that are regular across different tasks while being robust to the randomized elements, such as the background colour or the physics parameters in a manipulation task [46]. Domain randomization, however, limits the application of such systems to environments that fall within the randomized space. In vision-based system for example, unless the light-source is randomized in the simulation environment, it won't be able to work robustly in the real world [22].

In many applications where the attributes of the desired object and environment conditions are known a priori, end-to-end systems have been successfully deployed to map real-world images to actions [49]. However, the use of end-to-end systems for multi-task manipulation remains limited. For instance, a policy that is trained for grasping apples cannot be used to grasp an orange instead without necessitating further training or domain adaptation [15, 24]. We overcome this limitation by using a shared representation of manipulation tasks with different desired and obstacle objects.

More recently, the research on manipulation in clutter is witnessing a growing interest in image-based systems with top-down bin picking that acknowledge the necessity of accounting for physics interaction in cluttered environments. The work by Zeng et al. [62] approaches object picking from a bin using a custom designed gripper capable of push, grasp, and suction actions. Working with RGB-D images, they use several neural network models (one per action primitive) to evaluate pixel-wise affordances for the corresponding actions, then execute the action at the location and orientation of the highest affordance. Also using RGB-D images, Shome et al. [52] use a suction cap end-effector to perform one of three manipulation primitives, namely toppling, pulling, and pushing for bin packing tasks of cuboidal objects. For grasping in a tightly packed cluster of objects, Zeng et al. [61] learns synergetic push and grasp actions over pixel-wise action-value heat map to disperse the clutter then grasp one of the objects in the scene. These approaches, however, require specific high-level manipulation primitives to be defined. On-shot execution of a high-level primitive limits the robotic manipulator from dynamically correcting its behaviour in response to unexpected changes in the environment. Viereck et al. [58] addresses this problem by using a controller that continuously updates the target grasp pose during execution. The controller relies on a learned distance to grasp function that is evaluated over depth information. It remains that these approaches are object agnostic, i. e., no specific object to be manipulated can be specified apriori and it is left to the system to select an object that is feasible to grasp. In many real-world scenarios, it is desirable to be able to specify the target object such as for getting a milk bottle from the back of a cluttered fridge.

In environments with sparse reward functions, remarkable results have been achieved by combining RL based approaches with planning which compensates for the suboptimality of a policy with informed model-based conjectures. This combination is dominated by tree search planners to guide RL policy search [3, 53], mostly by the adoption of Monte Carlo Tree Search (MCTS) that uses Upper Confidence Bounds (UCB) to balance between state space exploration and rewards exploitation [27, 48]. Song et al. [54] apply this concept to physics-based manipulation domains for solving rearrangement tasks that necessitate a long sequence of actions. Albeit the use of neural networks that are recursively trained over previous iterations of the generated plans to speed up the search [8, 32], the computation cost associated with the transition function of physics models remains prohibitively expensive for this process to run in closed-loop in real-time. This is because, fundamentally, MCTS requires a large number of roll-outs for its estimates to become reliable.

A more viable alternative is the use of Model Predictive Control (MPC) and RHP like approaches to perform look-ahead planning in real-time. If the goal state is not within the horizon reach, a learned value function is used as a terminal cost at the horizon state in order to solve a finite-horizon approximation to a long or infinite horizon task. Kartal et al. [26] extend an Actor-Critic RL algorithm to predict the temporal closeness to terminal states. The temporal closeness is then used to enable limited-depth MCTS roll-outs with around a 100 roll-outs per action selection. Despite significant improvement over previous works, the number of performed roll-outs remains far above real-time performance for physics simulation. In a physics simulator, Tong et al. [57] propose training a policy network to generate a sequence of actions up to a certain horizon. Then using another value function trained network and an evolutionary algorithm, the sequence of actions is optimized and the first action of the optimized sequence is executed. Thananjeyan et al. [55] implements deep MPC over learned dynamics in a constrained environment. To ensure exploration without violating the constraints, generated trajectories are conditioned to where a plan exists for navigating back to a safe set. More recently, to address control problems under constraints such as state boundary, Mittal et al. [39] train Lyapunov neural network to be used as the terminal cost for a one-step horizon MPC with an imperfect forward model to improve on the control stability. Although the approaches stated so far are showing promising results and have a remi-

niscence to our work, some of their underlying assumptions, such as having access to a dense reward function or that the initial and goal state distributions forms a tight subset of the state space, are not applicable to physics-based manipulation in clutter tasks.

3. Framework

The framework is composed of two phases: first the heuristic learning phase which takes place in simulation, then the execution phase in which VisualRHP is interleaves the real-world actions with the physics simulator at run time. We will begin by describing the execution phase (Sec. 5) assuming learning has already taken place, and then characterize the learning phase that makes it possible (Sec. 6).

The execution phase consists of a closed-loop control scheme (Fig. 2-Bottom). It dynamically maps the state of the real world to the simulator, where an action is selected and then executed by the real robot. The control scheme cycle starts by processing multi-sensory data from the real world to produce a corresponding state in the physics simulator (Sec. 5.1). Then, in the simulator, VisualRHP performs a local look-ahead search by simulating multiple physics-based roll-outs up to a certain horizon. Each roll-out is evaluated by computing its expected return. In this process, a heuristic is required for (*i*) guiding the local search towards parts of the state space with high expected returns, (*ii*) and for estimating the expected return beyond the horizon state (Sec. 5.2). VisualRHP returns the *first* action of the best roll-out. Lastly, the selected action is resolved to the joint motion of the real robot (Sec. 5.3).

The learning phase consists of training a Deep Neural Network (DNN) to be used as the VisualRHP heuristic (Fig. 2-Top). The heuristic guides VisualRHP to produce a behaviour that seamlessly generalizes over different environment setups. The DNN in the discrete action space is trained to approximate the optimal action-value function, whereas the DNN in the continuous action space is trained to approximate the optimal policy as well as the value function of the learned policy. For both the discrete and continuous action space implementations, we use deep IL (Sec. 6.2) followed by deep RL (Sec. 6.3) to train the DNN.

4. Problem Formulation

We formalize the problem as a Markov Decision Process (MDP), represented as a tuple $M = \langle S, A, T, r, \gamma \rangle$ where S is the set of the environment states describing:

- the surface edges represented by a set of vertices: $edges = \{(x, y)^1, (x, y)^2, \dots\}$
- the location and radius of the circular target region: $tarReg = \{(x, y)^{\text{tarReg}}, rad^{\text{tarReg}}\}$ with $(x, y)^{\text{tarReg}}$ being within $edges$
- the Cartesian pose and gripper state of the robot end-effector: $rob = \{(x, y, \theta)^{\text{rob}}, \theta^{\text{grip}}\}$

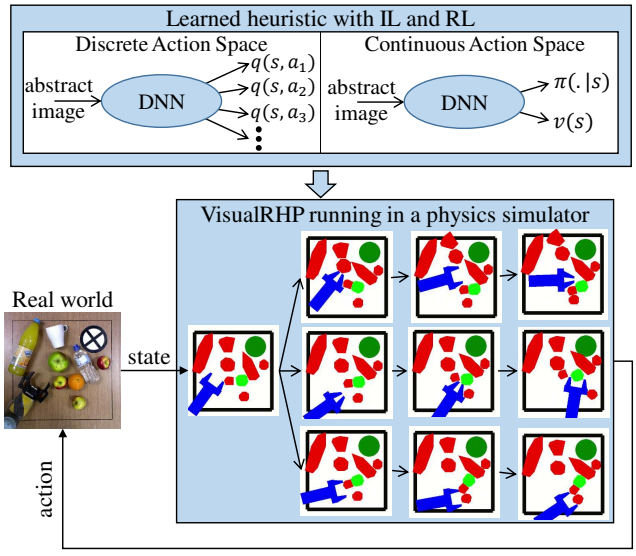


Fig. 2. Framework overview. Example with the orange fruit as the desired object.

- the arrangement of the desired object and the obstacle objects: $arr = \{(x, y, \theta)^{\text{desObj}}, (x, y, \theta)^2, \dots, (x, y, \theta)^m\}$
- the shape of the objects and the robot end-effector, represented by their vertices: $ver = \{ver^{\text{desObj}}, ver^2, \dots, ver^m, ver^{rob}\}$

where m is the number of objects in the scene; $tarReg$ and $desObj$ stand for target region and desired object, respectively. Therefore, a state s at time t is given by $s_t = [edges, tarReg, rob, arr, ver]$; A is the set of actions that the robot can execute for moving over a planar surface and for closing and opening the gripper. A can be either defined over a discrete or a continuous space; $T : S \times A \times S \to [0, 1]$ is the transition probability function, $r : S \times A \times S \to \mathbb{R}$ is the reward function; γ is the discount factor.

We denote as S_{val}, the set of *valid* states where all the of objects lie within the surface edges. The set of *invalid* states, S_{inval}, consists of the states where any of the objects is located outside of the surface edges. The task goal states, $S_g \subset S_{val}$, is identified by the arrangement, arr, where the desired object is in the target region, satisfying:

$$||(x, y)^{\text{desObj}} - (x, y)^{\text{tarReg}}|| \leqslant rad^{\text{tarReg}}.$$

Intuitively, it is expected from the robot to manipulate the desired object to the target region with the least number of actions without violating the surface edge constraints. The robot interacts with the environment following a policy $\pi(a|s)$, then the environment transitions to the next state s' and the robot receives a reward $r(s, a, s')$. In this work, we adopt a sparse reward function to avoid shaping exploration through reward engineering. The optimal policy $\pi^*(a|s)$ maximizes, at any instant t, the expected discounted sum of future rewards (called the *return*) $G_t = \sum_{k=t} \gamma^{k-t} r_{k+1}$, where $r_{k+1} = r(s_k, a_t, s_{k+1})$. Therefore, the optimal policy maximizes:

$$J(\pi) = \mathbb{E}_{s\sim d, a\sim\pi}[G_0|s_0 = s]. \quad (1)$$

where d is the initial state distribution and actions are sampled from π. The robot maximizes the expected return by manipulating the objects in the environment along a sequence of states $\langle s_t \rangle_{t=0}^{L}$ $s.t.$ $s_t \in S_{\text{val}}$ for t in $[0, L]$, where $L+1$ is the number of traversed states, from $s_0 \in S_{\text{val}}$ to $s_L \in S_{\text{g}}$.

The optimal policy can be approximated either indirectly with value iteration methods or directly with policy iteration methods. **In the discrete action space**, a DNN parametrized by a vector ψ, $q_\psi(s, a)$, is trained through value iteration to approximate the optimal action-value function:

$$q^*(s,a) = \int_{s' \in S} T(s,a,s')[r(s,a,s') + \gamma \max_{a'} q^*(s',a')]ds',$$

that is the expected return for taking action a at state s and proceeding following the optimal policy. The optimal policy can be derived by $\pi^*(a|s) = \max_a q^*(s,a)$. The *max* operator allows for learning the action-value function with off-policy methods, i. e., learning the value of the optimal policy independently of the learner's actions. **In the continuous action space**, the parameters of the policy can be directly learned with an Actor-Critic method similar to Advantage Actor-Critic (A2C). It uses value and policy iterations until it converges to the optimal policy $\pi^*(a|s)$. The performance of a θ-parametrized stochastic policy $\pi_\theta(a|s)$, i. e., the Actor, can be iteratively improved by evaluating its performance with respect to a ϕ-parametrized value function $v_\phi(s)$, i. e., the Critic. $\pi_\theta(a|s)$ models a k-dimensional multivariate Gaussian distribution $\mathcal{N}_\theta(\mu_{A_k}, \sigma_{A_k})$ from which actions can be sampled, and $v_\phi(s)$ approximates the expected sum of discounted rewards from following π_θ at state s: $v(s) = \mathbb{E}_{a\sim\pi_\theta}[G_t|s_t = s]$. It is common, in applications where the visual input is significant, for the value function and the policy to share some of their parameters, i. e., ϕ and θ, mostly the convolutional part of the DNN. The motivation behind shared parameters is that the low level features useful for estimating the value function could also be useful for modelling the policy and vice versa. Furthermore, optimizing the parameters of the value function and policy together acts as a regularizing element which leads to greater stability in the learning process [40, 51]. In the rest of this article, we will refer with θ to the parameters of the Actor-Critic DNN.

Instead of acting greedily w. r. t. the learned value function or policy, using them as heuristics for RHP mitigates inaccuracies in their learned approximations of the optimal behavior. The longer the RHP horizon is, the less the behavior depends on the learned approximations and vice versa.

We assume (*i*) a quasi-static physics model with limits on the velocity of the robot motion, (*ii*) full observability of the environment, and (*iii*) discrete time steps.

5. Execution Phase

In this section, we present a closed-loop control scheme (Fig. 2-Bottom) that balances real-time execution with long-term goal-oriented actions. First, we explain the mapping from the real-world to the simulator and the colour-labelled abstract image-based state representation (Sec. 5.1), and then we explain how VisualRHP uses a heuristic acting on the abstract state representation to suggest actions (Sec. 5.2). Once an action is chosen, it is resolved back to the robot joint space and executed by the real robot (Sec. 5.3).

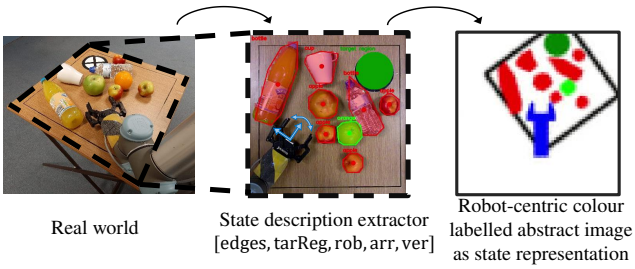


Fig. 3. Mapping the real-world state to a colour-labelled abstract image-based state representation for a task where the orange fruit is the desired object.

5.1. Mapping the real world to an abstract representation

Our mapping captures the state of the real world while leaving out information that is not relevant to the task, such as object texture, background colour, lighting sources, etc. As illustrated in Fig. 3-Middle, we apply *instance segmentation* on real-world images to detect the number *m*, arrangement *arr*, and the shape *ver* of the objects. This operation is performed using *Mask R-CNN* [19] which also identifies the type of each detected object. We identify the target region *tarReg* using simple template matching. We localize with forwards kinematics the end-effector pose over the planar Cartesian space and the gripper state *rob*. The simulator uses this information to create objects with the same contour shapes as in the real world. In this work, the shape of the end-effector and the surface edges are pre-loaded into the physics model as they are kept unchanged from one task to another[1].

The input to the DNN is in the form of an image rendered from the state of the physics simulator. The objects in the simulator are colour labelled based on their functionality. As in the example of Fig. 3-Right, the desired object is always of the same colour (light green), all other objects are of another common colour (red). The same applies to the end-effector (blue), the surface edges (black), the target region (dark green), and the scene background colour (white) across all task instances[2]. The colour labelling allows for the seamless transfer of manipulation skills to different desired objects. For example, any real-world object can be assigned the colour of the desired object and the DNN will treat it as such.

Furthermore, the abstract images are made robot centric, i. e., the image tracks the robot from a top-view perspective with constant offset (Fig. 3-right). The robot centric view

[1] If required, the detection of the shape of the end-effector and the surface edges can also be automated.

[2] Where required, additional information, such as objects where the quasi-static assumption does not apply, can also be captured by the colour labelling.

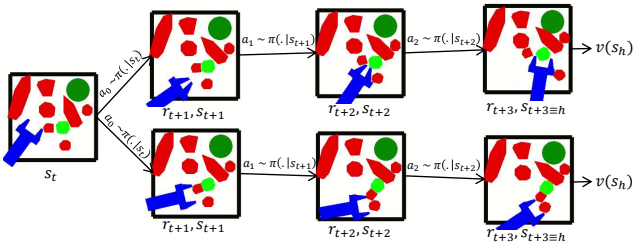


Fig. 4. VisualRHP example of $n = 2$ roll-outs with $h = 3$ horizon depth in continuous action space.

reduces the amount data required by the learning algorithm due to the symmetry of the scene when compared to a fixed view.

An equally important advantage of using this abstract representation is in reference to the size of the DNN architecture need to operate on it. For real-time operation, it is essential to have a small network to ensure fast inference time. The abstract representation allows for a much smaller convolutional part of the DNN to capture relevant features when compared to the convolutional architectures designed to handle real-world or realistically rendered images [7].

5.2. VisualRHP

VisualRHP can be seen as a combination of two processes. The first process consist of performing a local search starting from the current state of the simulator up to a certain horizon depth. However, an exhaustive search would scale badly with a horizon depth h and the size of the action set A, $\mathcal{O}(|A|^h)$ in the discrete case, and it would be inapplicable in the continuous case. Instead, as illustrated in an example in Fig.4, we run a small number of n roll-outs up to a short horizon of depth h while using the heuristic as a stochastic policy to orient the roll-outs' expansion towards directions with high return. The stochastic policy for the discrete action space is the soft-max of the action-value function:

$$P_\psi(a|s) = \frac{exp(q_\psi(s,a)/\tau)}{\Sigma_{a_i \in A} exp(q_\psi(s,a_i)/\tau)}, \tag{2}$$

where τ is the temperature parameter, while it is represented explicitly for continuous actions $a \sim \mathcal{N}_\theta(\mu_{A_k}, \sigma_{A_k})$. The stochastic policy would favour exploring actions that are more likely to lead to higher return.

The second process consists of computing the expected h-step return of a roll-out, using the first h rewards generated by the model and the expected return beyond the horizon state s_h. In discrete action space, the value of a horizon state s_h is computed as $v(s_h) = \max_a q_\psi(s_h, a)$, whereas in the continuous action space it is the output of the value head of the DNN: $v(s_h) = v_\theta(s_h)$. The return of a roll-out is therefore computed as:

$$R_{t:h} = r_1 + \gamma r_2 + \ldots + \gamma^{h-1} r_h + \gamma^h v(s_h). \tag{3}$$

The VisualRHP algorithm, detailed in Alg.1, returns the *first* action of the roll-out that obtained the highest return $R_{t:h}$.

Algorithm 1: VisualRHP(s_{cur}, n, h)

Input: Current state s_{cur}, No. rollouts n, horizon h
Output: action a

```
RolloutsReturn ← [ ]; FirstAction ← [ ]
for n_i = 1,2, ..., n do
    setSimulatorTo(s_cur)
    s ← s_cur; R ← 0
    for h_i = 1,2, ..., h do
        a ∼ π(.|s)
        if h_i is 1 then
            FirstAction.append(a)
        s ← simulatePhysics(s, a)
        r ← receiveRewards
        R ← R + γ^(h_i − 1) r
        if isTerminal(s) then
            break
    if not isTerminal(s) then
        R ← R + γ^h v(s)
    RolloutsReturn.append(R)
return FirstAction[argmax(RolloutsReturn)]
```

In this algorithm, the heuristic plays two roles: it informs the search through sampling, and as an approximation of the rewards that are not sampled from the model.

5.3. From the simulator to the real world

Although VisualRHP can be extended to arbitrary dimensions of the action space, we limit our implementation to actions parallel to the manipulation surface in the planar Cartesian space of the robot end-effector. It is safe to assume that a sequence of actions performed over a small planar Cartesian space can be resolved to the joint motion of a redundant manipulator. Kinematic singularities can be avoided using an inverse kinematics solver based on non-linear optimization [33]. Therefore, the action returned by VisualRHP is resolved to the robot joint motion and executed in the real world.

6. Learning Phase

As described in Sec. 5, the performance of VisualRHP is dictated by the quality of its heuristic. We are interested in approximating the optimal heuristic for VisualRHP, whether in the form of the optimal action-value function or in the form of the optimal policy and value function for the discrete and continuous actions space, respectively. We formulate the heuristic learning as an RL problem where the robot is trained in simulation to maximize the return (Eq. 1).

Training a randomly seeded RL algorithm in a cluttered environment under edge constrains is unlikely to converge to a good solution, as transition samples leading to the goal will not be observed enough many times. For this reason, we use a planner as a starting point for the search. The DNN is jump-started with IL from demonstrations generated by a probabilistically complete sampling-based planner. With enough knowledge captured from demonstrations, RL would

only require a relatively small number of transition samples to refine the robot's behaviour. Therefore, we divided the heuristic learning process into three sequential steps: (*i*) generating demonstrations, (*ii*) IL, and (*iii*) RL. We detail each of these steps in discrete and continuous action spaces.

6.1. Generating demonstrations

Sampling-based planners provide a probabilistically complete tool to solve complex planning problems in high dimensional state and action spaces without requiring a handcrafted or domain-dependent heuristic. In particular, Kino-dynamic planners are one family of the sampling-based Rapidly exploring Random Trees planners (Kino-dynamic RRT), specific for solving planning problems that involve dynamic interactions. We implement a discrete and a continuous version of the state-of-the-art Kino-dynamic planner [18] used for solving planning problems on physics-based manipulation in clutter.

We generate P task instances. Each task instance is initialized with random environment setups. This includes the location of the target region, the initial arrangement, and the shape and number of objects. Then, for each task instance p, we run the Kino-dynamic planner to generate a solution of the form $\langle a^p_0, \dots, a^p_{L-1} \rangle$ with state sequence $\langle s^p_0, \dots, s^p_L \rangle$.

6.2. Imitation Learning

In this section, we show how to use the generated plans as demonstrations to train the DNN to reproduce the behaviour of the Kino-dynamic RRT.

6.2.1. IL in Discrete Action Space

In the discrete action space, the goal is to learn the action-value function from the transition samples observed in the demonstrations. The actions never selected by the planner, which do not appear in the demonstrations, must be penalized by receiving a lower value with respect to the selected actions.

We train the DNN to predict the value of the actions selected by the planner at the visited states by minimizing the mean squared error w. r. t. the *Monte Carlo* target:

$$q^{tar}(s^p_l, a^p_l) = \sum_{k=0}^{L-l-1} \gamma^k r_{l+k+1}, \tag{4}$$

where p stands for the index of the demonstration and l for the index of the state-action pair in that demonstration.

While the DNN trained as above learns to predict the action-values for the actions selected along the visited states, the values predicted by the DNN for actions that have not been selected by the planner along the visited states can be arbitrary. As a result of function approximation, these actions must have a value even though they do not appear in the training set. The value can converge to an arbitrary number, determined by the effect of the target action value of the traversed state-action transitions. A possible undesirable effect is that the values of the actions not selected by the planner can be higher than the selected one. This can later cause an action that was not favoured by the Kino-dynamic planner to look more favourable to VisualRHP that uses the action-value function as a heuristic (Eq. 2).

To counteract this phenomenon, we use for the unselected actions a target value that is lower than the value of the selected actions. The minimum allowed difference between the value of the selected action and the other actions is referred to as the *value margin* function (λ) [21, 47]. We propose a definition of λ driven by the observation that, in the domain of planar manipulation tasks, a mistake is in most cases not irreparable, but can be overcome through a number of η additional actions of fixed cost $r_{cost} > 0$, such that, $\lambda = \sum_{k=1}^{\eta} \gamma^{L-l-1+k} r_{cost}$. This definition scales λ down the further away s_l is from the final state in a demonstration. Hence, we set the action-value target of the unselected actions at visited states to:

$$q^{tar}(s^p_l, a^p_u) = \begin{cases} q^{tar}(s^p_l, a^p_l) - \lambda, & \text{if } q_\psi(s^p_l, a^p_u) \geq q^{tar}(s^p_l, a^p_l) - \lambda \\ q_\psi(s^p_l, a^p_u), & \text{otherwise,} \end{cases} \tag{5}$$

where $a_u \in A \setminus \{a_l\}$ is an unselected action. If the value that the DNN converges to does not favour an unselected action then we leave it unchanged. Lastly, we add an L_2 regularization term to avoid over-fitting on the demonstrations.

6.2.2. IL in Continuous Action Space

The value head of the θ-parameterized DNN is trained to estimate the future sum of discounted rewards that are expected to be collected if RRT were to be engaged at the current state. The update target is similar to Eq. 4 but it is computed over a state instead of a state-action pair $v^{tar}(s^p_l) = \sum_{k=0}^{L-l-1} \gamma^k r_{l+k+1}$. Furthermore, we train the policy head of the θ-parameterized DNN to estimate the action distribution over states visited in the demonstrations while penalizing high entropy distributions. One way this can be achieved is by minimizing the loss function that combines the policy and the value function:

$$\begin{aligned} \mathcal{L}_{\text{il}}(\theta) = \mathbb{E}_{s^p_l, a^p_l}[& - \Upsilon \, log \pi_\theta(a^p_l | s^p_l) \\ & + c_1 \, (v^{tar}(s^p_l) - v_\theta(s^p_l))^2 \\ & + c_2 \, H(\pi_\theta(.|s^p_l))], \end{aligned} \tag{6}$$

where c_1 and c_2 are hyper-parameters. Υ is a positive constant, such that the first term on the right of Eq. 6 increases the likelihood of action a^p_l at state s^p_l. The second term updates the value estimate w. r. t. the *Monte Carlo* target, and the last term is an entropy penalty added to reduce the probability of unselected actions at visited states.

We experimented with DNNs of different sizes and expressive power in both the discrete and continuous action cases, but none could reliably represent the behaviour of the planner over a large number of task instances. As a consequence, we show in the next section how the information compiled in the DNN can be further optimized to play a valuable role when used as the VisualRHP heuristic.

6.3. Reinforcement Learning

So far, the knowledge encapsulated in the DNN has two shortcomings: first, the plans generated by the Kino-dynamic planner are, in general, sub-optimal; and second, some information is lost in the approximation by the DNN, with consequent performance degradation with respect to the Kino-dynamic planner. To overcome these problems, we use RL to (*i*) improve the DNN to better estimate the return of the optimal policy and/or to better estimate the optimal policy and to (*ii*) learn the value of the not experienced state-action transitions.

6.3.1. $\mathcal{E}$-RHP as the RL Policy in Discrete Action Space

Operating in the discrete action space allows for a straightforward implementation of an off-policy RL algorithm. Off-policy makes it possible to leverage RHP in the exploration policy of the RL algorithm to exploit actions that are more likely to lead to the goal. We implement the Deep Q-Learning algorithm (DQN [41]) with RHP-based exploration policy.

We initialize the DNN with the IL trained ψ parameters. We also initialize a large buffer D^{replay} with the demonstration transition samples. Further, we formulate a novel exploration policy, that we call ϵ-RHP, which selects a random action with probability ϵ and with probability $1-\epsilon$ the policy queries RHP for an action. We found that focusing the search towards the goal, by augmenting the RL policy with RHP, reduces the chances of the action-value function from diverging which is a common problem in RL when used in conjunction with neural networks as a function approximator. The robot uses ϵ-RHP to collect transition samples over task instances initialized with random environment setups. Throughout the data collection process, the robot stores the newly collected transition samples in the buffer D^{replay}. The old samples in the buffer get gradually replaced by new ones that are collected with ϵ-RHP. When enough new transition samples are collected, the DNN is updated by running a batch optimization over randomly sampled transitions from D^{replay}. The loss function over a batch $B = \{\langle s_i, a_i, r_i, s'_i\rangle_{i=1}^{M}\}$ of M transition samples is defined as:

$$\mathcal{L}_q(\psi) = \frac{1}{M}\sum_{i=1}^{M}(r_i + \gamma \max_{a'} q_\psi(s'_i, a') - q_\psi(s_i, a_i))^2. \quad (7)$$

An L_2 regularization loss is also added on the network parameters.

6.3.2. Critic Correction Conditioned Policy Optimization (C3PO) extension for A2C in Continuous Action Space

In the continuous action space, we propose reformulating the loss function for an A2C style algorithm. The motivation is to improve on the stability of existing algorithms that use shared neural network parameters and are sensitive to changes in the policy, more specifically to avoid *catastrophic forgetting*.

One way of implementing A2C with shared parameters is by (*i*) running multiple simulation environments in parallel, each with random task parameterization and with the same copy of the DNN. In each environment, the robot is controlled by π_θ. (*ii*) Once all the transition samples in D^{replay} are replaced with the ones collected with π_θ, the parameters θ of the DNN are stored as θ_{old}. (*iii*) Then, the policy and the value function are updated together by minimizing in batches $B = \{\langle s_i, a_i, r_i, s'_i\rangle_{i=1}^{M}\}$ the loss function w. r. t. θ:

$$\begin{aligned}\mathcal{L}_{\text{actor-critic}}(\theta) = \frac{1}{M}\sum_{i=1}^{M} & - Adv(s_i, s'_i, a_i)\frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)} \\ & + c_3\,(r(s_i, a_i) + \gamma v_{\theta_{old}}(s'_i) - v_\theta(s_i))^2 \\ & - c_4\, H(\pi_\theta(.|s_i)), \end{aligned} \quad (8)$$

where c_3 and c_4 are hyper-parameters, Adv is the advantage function estimate:

$$Adv(s_i, s'_i, a_i) = r(s_i, a_i) + \gamma v_{\theta_{old}}(s'_i) - v_{\theta_{old}}(s_i), \quad (9)$$

computed w. r. t. the learned baseline, namely the value function $v_{\theta_{old}}$ [40]. H is added to encourage exploration by limiting the premature convergence to a sub-optimal policy [40].

The problem with this formulation of the loss function is that, in the policy update component of Eq. 8, the advantage function Adv is using a baseline $v_{\theta_{old}}$ that has not yet been updated to capture the value of the policy $\pi_{\theta_{old}}$ used to collect the samples in D^{replay}. This means that the baseline for updating the policy is always one step behind the policy used to collect the data. This often goes unnoticed as the policy updates are usually bounded to small changes in policy gradient based algorithms. For example, PPO uses the *clip* function on the ratio $\frac{\pi_\theta}{\pi_{\theta_{old}}}$ [51], whereas TROP imposes a constraint on the KL-divergence between the new policy and the old policy [50]. However, in environments with non-linear and non-continuous dynamics, such as the case in cluttered environments, even a very small change in the policy can possibly entail a drastic change in the value function cascading into what is know as *catastrophic forgetting*.

To overcome this problem, we propose updating $v_{\theta_{old}}$ prior to the optimization step of Eq. 8, i. e., before using $v_{\theta_{old}}$ as a critic in Adv. The baseline is updated to improve the estimate of the value of the policy used to collect the latest round of data, while also refraining from causing a change to the action distribution of this policy. This is achieved by first doing an update of the value function w. r. t. θ:

$$\begin{aligned}\mathcal{L}_{\text{baseline}}(\theta) = \frac{1}{M}\sum_{i=1}^{M} & \; c_5\,(r(s_i, a_i) + \gamma v_{\theta_{old}}(s'_i) - v_\theta(s_i))^2 \\ & + D_{KL}(\,\pi_{\theta_{old}}(.|s_i)\;||\;\pi_\theta(.|s_i)\,), \end{aligned} \quad (10)$$

where c_5 is a hyper-parameter. The first term on the right updates the value function of the policy used to collect the transition samples. Since the policy and the value function share the same body of the DNN and updating one perturbs

Algorithm 2: Condition Critic Correction Policy Optimization (C3PO) extended A2C

$D^{replay} \leftarrow [\,]$
for *iteration = 1,2, …* **do**
  while $|D^{replay}| < M$ **do**
    for *actor = 1,2, …, N* **do**
      Generate random task instance
      Run $episode$ with policy π_θ
      D^{replay}.append($\langle s_i, a_i, r_i, s'_i \rangle_{i=0}^{L-1}$)
  $\theta_{old} \leftarrow \theta$
  Optimize $\mathcal{L}_{\text{baseline}}$ w. r. t. θ (Eq. 10)
  $\theta_{old} \leftarrow \theta$
  Optimize $\mathcal{L}_{\text{actor-critic}}$ w. r. t. θ (Eq. 8)
  $D^{replay} \leftarrow [\,]$

the other, the second term on the right penalizes the KL-divergence between the action distribution of the policy used to collect the data $\pi_{\theta_{old}}$ and any resulting change in the action distribution of π_θ that might be induced by the update of the value function v_θ. This procedure, which we call Critic Correction Conditioned Policy Optimization (C3PO), is outlined in Algorithm 2. This algorithm follows the same structure of A2C algorithms with the addition of the $\mathcal{L}_{\text{baseline}}$ optimization step. Hence, C3PO can be used as an extension to state-of-the-art A2C algorithms with shared neural network parameters. Learning a policy and value function to act as a heuristic for VisualRHP is also possible with off-policy RL algorithms, such as TD3 [16] and SAC [17]. However, since sample efficiency is not a key factor, as the DNN is first optimized with IL, we use C3PO on top of PPO as it is a relatively stable and robust to hyper-parameter algorithm.

7. Experimental Setup and Implementations

We evaluate the performance of the proposed VisualRHP in achieving *efficient* and *generalizable* planar manipulation using prehensile and non-prehensile actions with emphasis on real-world and real-time applications. We run a series of experiments conducted in simulation and on the real robot to evaluate and validate VisualRHP. The focus of the experiments are: (*i*) to evaluate the performance contribution of each of the main elements of VisualRHP in discrete and continuous action spaces with respect to state-of-the-art alternatives (Sec. 8.1), (*ii*) to assess the algorithms' robustness to un-modelled dynamics for real-world applications compared to open-loop execution (Sec. 8.2), (*iii*) to evaluate if the acquired behaviour learned transferable skills to different real-world manipulation environment (Sec. 9).

During the experiments, we varied the environment parameters introduced in Sec. 4. The environment consists of the target region, the end-effector of the robot arm, one desired object, and $m-1$ obstacles[3]. We trained the DNN on task instances with different clutter densities, ranging from $m = 1$ object, i. e., only the desired object, to $m = 7$ objects. The shape of an object is randomly selected from a pool of polygons with a random number of vertices centred around the polygon centre of mass. Some of the objects are too large to fit within the gripper fingers to be grasped, whereas others are small enough to be grasped with force closure from any approach angle, and some others are directionally graspable. The location of the target region is sampled from a uniform distribution over the manipulation surface. With the aim of only describing the task objective, the reward function is set to $r = -1$ per action and $r = -50$ if an object is dropped outside of the surface edges. The negative reward per action encourages the robot to solve the problem with as few actions as possible.

7.1. Action spaces

In our simulation environment, we modelled the world in the Box2D physics simulator [9]. The robot motion is driven by a PD controller, where an action represents a target velocity vector to be maintained for a fixed amount of time.

Discrete action space (*das*): We define 8 actions: four along cardinal directions that achieve a 5 cm translation in any of these directions, two 30^o rotational actions (CW and CCW), and the last two are for closing and opening the gripper.

Continuous action space (*cas*): The action distribution is modelled by a 4-dimensional Gaussian distribution. The first three correspond to the longitudinal, lateral, and rotational directions along the robot end-effector. The means of the Gaussian distribution are bounded such that the induced translation and rotational step falls within -10 cm and 10 cm, and -50^o and 50^o, respectively. The fourth dimension corresponds to closing and opening the gripper.

7.2. Network architecture

It is important to design a neural network architecture with an inference time small enough to query RHP multiple times per action selection and still return an action in near real-time. We built similar feedforward neural network architectures for the discrete and continuous action spaces, using the TensorFlow library [1], with the main difference being the output heads.

Discrete action space: The DNN modelling the action-value function is composed of a CNN part connected to dense layers. The input to the CNN is a $64 \times 64 \times 3$ image. The CNN consists of 3 sequences of: 2 coordConv appended channels[4], 2D convolution, normalization, and max-pooling layers. We use 3×3 kernel sizes and 8, 8, 16 filters for the convolution layers, respectively. The output of the CNN is flattened into a 256 feature vector. The input to the dense layers is a feature vector that concatenates the output of the CNN and a binary value corresponding to the gripper state. The dense layers are composed of 3 fully connected layers. The first 2 layers have 128 neurons each. We use leaky ReLu as activation function all throughout the network. The out-

[3]The surface edge dimensions are 50×50 cm, the target region has a radius of 7 cm, and the objects and robot density is 1 kg/m^2 with 0.3 as the friction coefficient. The robot dimensions are modeled after the *Robotiq 2F-85* gripper.

[4]The coordConv helps in capturing translation invariant features [37].

put layer consists of 8 neurons, one per action, with linear activation functions.

Continuous action space: The architecture of the CNN and the input vector to the dense layers are the same as in the network for the discrete action space. The CNN is followed by two shared fully connected dense layers of 256 and 128 neurons. The value head has a fully connected layer of 64 neurons with leaky ReLu activation function. It is followed by a single neuron output layer for the value function with a linear activation function. The Policy head has a fully connected layer of 128 neurons with leaky ReLu activation function. The output layer is a fully connected layer with 8 neurons modelling the mean and standard deviation parameters of a 4-dimensional Gaussian that is representing the policy. The 4 outputs corresponding to the means have a hyperbolic tangent (tanh) as activation function, and the other 4 outputs corresponding to standard deviations have a sigmoid activation function[5].

The inference time over either of the DNNs is measured to be around 0.003 $seconds$ running on an Intel Xeon E5-26650 computer equipped with an NVIDIA Quadro P6000 GPU card.

7.3. Evaluation metrics

Data for each experiment is collected over 300 runs. Unless otherwise specified, the performance is evaluated in simulation with respect to three metrics:

- **Success rate** represents the percentage of the successfully completed tasks. We consider a task to be successfully completed when the desired object is moved to the target region in under 50 actions without having any of the objects falling off the surface.
- **Action efficiency** is a relative measure in view of the scene complexity represented by the clutter density. It is calculated as $\frac{number\ of\ objects\ in\ the\ scene}{number\ of\ actions\ until\ completion}$. The higher the ratio of an approach, the more efficient it is.
- **Average execution time per run** is computed as the number of actions times the average time required for selecting an action, which can change for different planners, and the time required to roll the physics in the simulator.

7.4. Training procedure

We collected, for each of the discrete and continuous action space approaches, up to $P = 14,000$ demonstrations over random task instances using the Kino-dynamic RRT planner. We train the DNNs over these demonstrations as detailed in Sec.6.2 on IL. For the **discrete action space** we use the *value margin* parameters of $\eta = 3$ and $r_{cost} = 1$. For the **continuous action space** we set the hyper-parameters to $c_1 = 0.7$, $c_2 = 0.01$, and $\Upsilon = 0.5$. We use a learning rate of 0.0001 for the training process in both action spaces, a discount factor of $\gamma = 0.995$, and a batch size of $M = 2000$.

To ensure that enough demonstrations were collected for IL, we show in Fig. 5 how the success rate changes when increasing the number of available demonstrations. The plots show the average success rate with 90% confidence interval as measured over 15 batches of 20 random task instances. As expected, the graph shows an increasing trend w. r. t. the number of available demonstrations. After reaching a P of around 8000, the success rate starts to plateau at around 60% and 70% for the discrete and continuous action spaces, respectively. We also report a remarkably low action efficiency of 0.15 and 0.20 for the discrete and continuous action spaces, respectively.

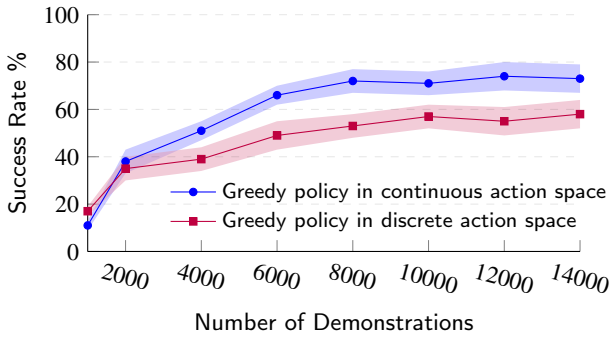


Fig. 5. The effect of the available number of demonstrations on the performance of the learned behaviour.

Next, the DNNs of both action spaces are further trained with RL. We keep the same batch size of $M = 2000$ and the discount factor of $\gamma = 0.995$. We decrease the learning rate to 0.00001. In the **discrete action space**, we use an ϵ-RHP with $\epsilon = 0.2$ and RHP with $n = 3$ and $h = 3$. We use a replay buffer D^{replay} that can fit 500000 transition samples. The transition samples are collected from 10 agents running in parallel. We run 3 optimization epochs after every 2000 newly collected transition samples. In the **continuous action space**, we use C3PO in conjunction with PPO. We set the PPO *clip* value to 0.075 and the hyper-parameters to $c_3 = 0.7$, $c_4 = 0.1$, and $c_5 = 0.35$. We set the size of the replay buffer D^{replay} to 10000 transition samples. We also run 10 environments in parallel. For the optimization step, we run 20 epochs over $\mathcal{L}^{baseline}$ and 15 epochs over $\mathcal{L}^{actor-critic}$.

7.5. VisualRHP and baselines

We conducted an ablation study to assess how each element in our proposed approach affects the final performance. We looked at the effect of the image-based abstract representation, the use of a learned heuristic, and the integration of the physics-based look-ahead planning in the control strategy. Accordingly, in addition to VisualRHP, we composed four corresponding baseline methods. We use $n \times h$ in the superscript of a method's name to denote the RHP parameters (e. g., $VisualRHP^{3\times3}$). All baseline methods are trained

[5] It is common in the continuous action space literature to use a linear activation function to model the standard deviations, however, we found that a tanh function makes it more likely for the RL algorithm to converge without having any noticeable downside on the exploration.

with the same procedure as ours unless otherwise specified:
VisualRHP: We ran our implementation in four formats. Two in the discrete action space and two in the continuous action space. For each of the action spaces, there is one version that uses $n = 3$ roll-outs of $h = 3$ horizon depth, and another one that uses $n = 6$ and $h = 6$.
Cartesian Pose Baseline (CartesianRHP): Instead of using abstract images for the state representation, *CartesianRHP* uses the relative Cartesian poses of the objects and the target region with respect to the end-effector, and the absolute Cartesian pose of the end-effector and a binary gripper state. The discrete and continuous action space versions of *CartesianRHP* use the same DNN architectures. We ran *CartesianRHP* in conjunction with RHP $n = 3$ and $h = 3$. The DNN architecture has an inherent limitation. It can only be trained on a specific number of objects and can not generalize to arbitrary clutter densities. Adding or removing objects, i. e., changing the size of the input layer, requires the use of a different DNN architecture. Hence, *CartesianRHP* requires the training of multiple DNNs, each designed to operate on a specific number of objects. This baseline is inspired by the one used in [6].
Handcrafted Heuristic Baseline (CraftedHeuristicRHP): We ask the question of whether the problem can still be solved in closed-loop with a handcrafted heuristic to estimate the cost-to-go from a horizon state to the goal, rather than the learned one. Hence, *CraftedHeuristicRHP* implements RHP with a handcrafted heuristic. We found that *CraftedHeuristicRHP* performs best with $n = 8$ random roll-outs of depth $h = 4$ by sampling random actions from the discrete or continuous action spaces. The cost-to-go function is a weighted sum of the Euclidean distance and the angular displacement between the robot and the desired object, the Euclidean distance between the target region and the desired object. It also includes a term that encourages the alignment between robot, desired object, and target region, with increasing emphasis on the robot facing the target region once it is positioned behind the desired object. A penalty term is added to dropping any of the objects outside the surface edges. The weights, balancing these different components, were empirically optimized to favour a behaviour where the robot would first approach the desired object from the back, then pushes it towards the target region. The use of a handcrafted cost function for manipulation in clutter is reminiscent to trajectory optimization-based approaches such as in [2, 45].
Greedy Baseline (Greedy): Almost ubiquitously, an RL trained robot for manipulation tasks would act greedily at execution time on the learned policy without running look-ahead planning [61, 49, 62]. In this work we argued that it is hard for a greedy policy to accurately anticipate how the environment will unfold under complex interaction dynamics, especially in an environment rich with physical collisions. *Greedy* challenges this claim by running a greedy policy on the trained DNN. Action selection is based solely on the current state as observed in the abstract image representation. In the discrete action space, the action with the highest value estimate is selected. In the continuous actions space, the action vector is set to the mean vector of the policy distribution as outputted by the policy head of the DNN. Therefore, the simulator at execution time is only used to render the abstract images on which the greedy policy acts. The most similar state-of-the-art approach to this baseline is [60].
Kino-dynamic RRT Baseline (K.RRT): All the previous baseline control strategies run in closed-loop. As an alternative, we used an open-loop sampling-based planner, namely the kino-dynamic RRT introduced in Sec. 6.1. A computation time limit of 3 minutes was imposed on *K.RRT* before declaring a failure. In the discrete action space, the planner has access to the 8 discrete actions per state. In the continuous actions space, the planner can sample up to 8 random actions per state.

We report that we omit the following two baselines from the results, as even after extensive systematic hyper-parameter tuning, we did not succeed in getting them to converge to a satisfactory behaviour:

- DNN parameters were randomly initialized and then solely trained with RL (i. e., without IL). This was repeated for several initialization trials. The policy often converged to a behaviour that drives the robot to shove the objects by the side of the gripper towards the target region without much consideration to the surface edges constraints, often causing objects to drop outside of it.

- A baseline wherein the RL part of our training procedure, in the continuous action space, uses PPO without the additional C3PO optimization step of $\mathcal{L}^{baseline}$. The original version of PPO resulted in the DNN diverging causing the loss of the acquired knowledge from the IL part. Fig. 6 shows a comparison of the success rate during the RL training process between PPO and C3PO with PPO. After every 10000 newly collected transition samples, 20 optimization epochs are performed. Each data point is averaged over 5 RL trials with 90% confidence interval. Both learning algorithms are evaluated with the greedy policy.

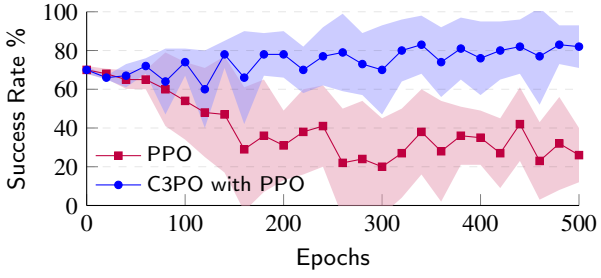
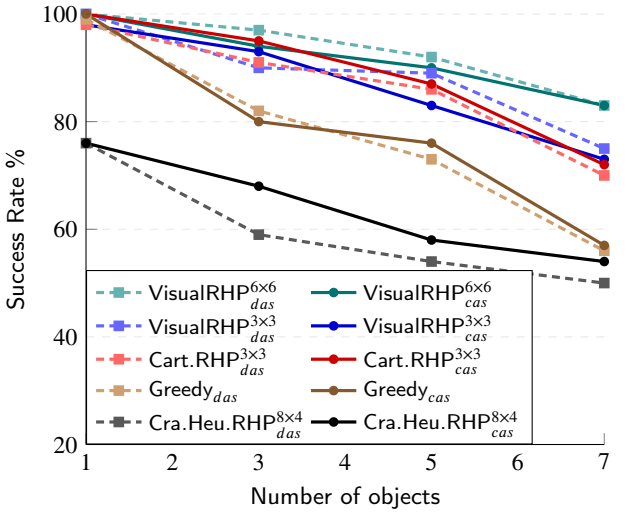
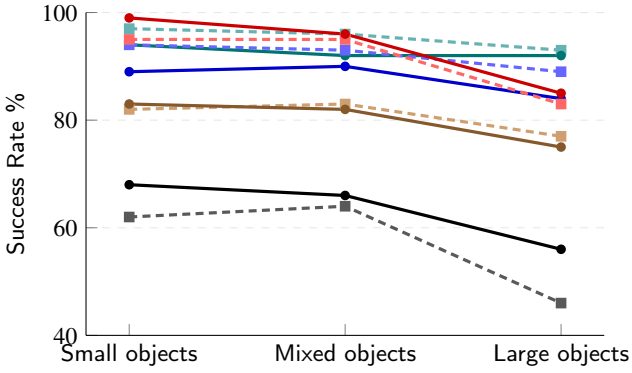


Fig. 6. The effect of the C3PO optimization step on the learning stability.

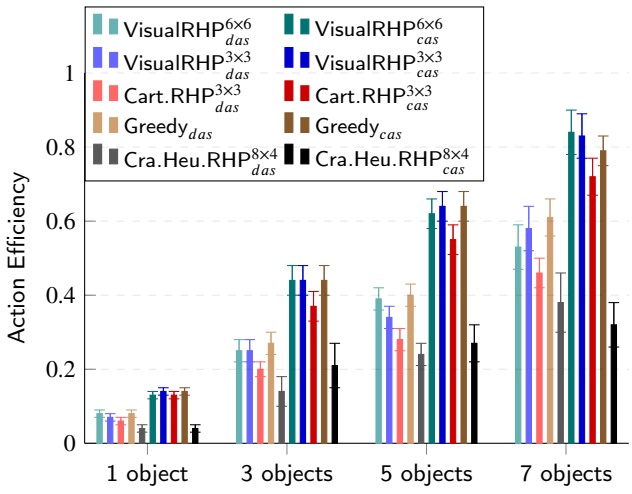


(a) Success rate w. r. t. clutter density.

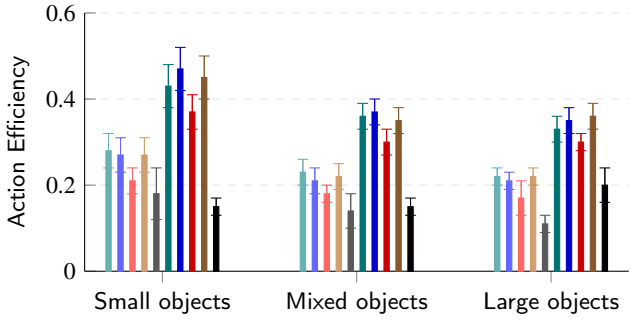


(b) Success rate w. r. t. to objects sizes.

Fig. 7. Performance in different environment setups.



(a) Action efficiency w. r. t. clutter density.



(b) Action efficiency w. r. t. object sizes.

Fig. 8. Action efficiency in different environment setups.

8. Simulation Results and Discussion

In this section we present the results and discuss the implications of the simulation experiments.

8.1. Performance in different environment setups

The first group of simulated experiments looks at the success rate, the action efficiency, and the average execution time per run in environments of different levels of difficulty. We consider the effect of changing clutter densities, ranging from 1 object, i. e., only the desired object without clutter, to 7 objects on the surface. We also examine the performance in environments with random number of objects (up to 7) but of different average sizes (small, mixed, and large) relative to the dimensions of the gripper. We expect that the shape of small objects is less significant to the manipulation task compared to large objects.

The success rate results are reported in Fig. 7a and in Fig. 7b. The plots show that *VisualRHP* and *CartesianRHP* outperform the other two baselines, with a slight advantage for running RHP with higher number and deeper roll-outs. For a low clutter density, all approaches show high success rate. Not surprisingly, increasing the clutter density causes a drop in the success rate across all baselines as it becomes much more likely for objects to fall off the edges or for the robot not to find its way through the clutter. *Greedy* suffers from the sharpest drop with respect to the number of objects. We also observe in Fig. 7b that large objects seem to be slightly more difficult to manipulate as reflected in a decrease in the success rate. RHP, on the other hand, is more robust to the increase in object sizes. This result can be attributed to the fact that with higher clutter density and/or objects sizes, more physical interactions are involved. The decision making process must account for these interactions over the short and the long term to avoid irreparable arrangements. In this sense, RHP compensates for the DNN deficiency to anticipate how the environment will unfold under a sequence of actions. Operating in continuous versus discrete action space has no significant effect on the success rate.

Looking at the action efficiency results in Fig. 8a and in Fig. 8b reveals more insight on the difference between operating in continuous and discrete actions spaces. There is a clear advantage of operating in the continuous action space. It consistently scores higher in all baselines. This is because the policy has finer control over the positioning of the robot.

We note that RL caused a significant increase in the action efficiency compared to the IL action efficiency. In the

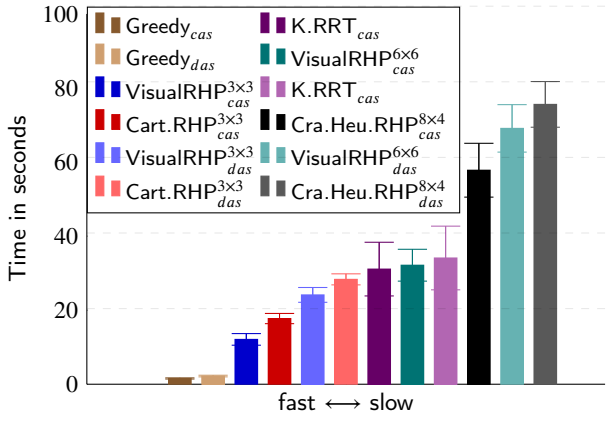


Fig. 9. Average planning and execution time.

mixed object experiment, the action efficiency of the greedy policies increased from 0.15 to 0.22 and from 0.20 to 0.35 in the discrete and continuous action spaces, respectively.

Furthermore, using RHP with higher number and deeper roll-outs (*VisualRHP*$^{6\times6}$) does not have a noticeable action efficiency advantage over using less (*VisualRHP*$^{3\times3}$) or no roll-outs (*Greedy*). The action efficiency results also show that although *CartesianRHP*$^{3\times3}$'s success rate is on par with *VisualRHP*$^{3\times3}$, *CartesianRHP*$^{3\times3}$ slightly but consistently scores lower on the action efficiency compared to all the approaches that rely on the abstract image-based representation. This confirms our intuition on the use of engineered features for the state representation. *CartesianRHP*$^{3\times3}$ always converged to a behaviour where the robot would approach the desired object from the back and push it towards the target region. Although robust to the variation in shape of the objects, each DNN of *CartesianRHP*$^{3\times3}$, trained over a specific number of objects, resulted in a behaviour that requires more actions for solving the task when compared to a behaviour where the robot can actually leverage the shape of the objects in order, for instance, to grasp the desired object and move it to the target region.

The importance of the action efficiency metric is reflected in the average execution time per task as presented in Fig. 9. We see that the average execution time per task is always lower in the continuous action space. This can be explained as a direct consequence of the action efficiency[6]. In addition, the average execution time is also affected by the number and depth of the RHP roll-outs. For instance, *VisualRHP*$^{6\times6}$ and *CraftedHeuristicRHP*$^{8\times4}$ simulate 6×6 and 8×4 actions before returning an answer. This places them at the slowest end of the spectrum. In contrast, *Greedy* does not need to perform any roll-out and exhibit the fastest execution time, albeit with a trade-off on the success rate. We also included in this figure the results for kino-dynamic RRT. It stands in the middle of the rank, but as we will see in the next section, it might not be the best suited for these kinds of domains.

8.2. Robustness to un-modelled dynamics

Following the results of the first group of simulated experiments, we identify that *VisualRHP*$^{3\times3}_{cas}$, *CartesianRHP*$^{3\times3}_{cas}$, *Greedy*$_{cas}$, and *K.RRT*$_{cas}$ are potentially suitable for near real-time applications in the real world. They offer a reasonable balance between computation time and success rate.

As a way of gauging how these approaches cope with dynamics that are different than the one they were trained for, we ran a second group of simulated experiments where we compared them against different levels of artificially injected noise on the physics and geometric parameters at evaluation time. Their performance under such conditions is a way of estimating the robustness of each policy, and approximating how a policy would perform under real world uncertainty. The results are presented in Fig 10. The different line styles correspond to the different noise levels. Each data point corresponds to 300 runs initialized with random target location, arrangement, and objects shapes. We injected noise into the shape, friction, and density parameters of the objects. During evaluation, the noise was sampled from a Gaussian distribution centered around the mean value of the parameters used in the training (and planning in the kino-dynamic planner case)[7].

K.RRT with no noise shows a high success rate. The few cases where it failed were due to the imposed time limit. Nevertheless, the decreasing performance with higher noise and the relatively high computation time, which also limits real-time re-planning, confirms the limitation of using open-loop planning in execution. In general, this indicates that this type of planning is favorable when a high-fidelity model and high-processing power are available. *Greedy*$_{cas}$ preforms remarkably well even with high noise but only for when there is one object in the scene. This can explain the wide spread use in literature of the greedy policy for pushing an object on a surface in the real world (see Sec. 2). It has real-time reactive behavior but it fails to cope with high clutter environment with unknown dynamics.

When looking at *CartesianRHP*$^{3\times3}_{cas}$ in high noise environments, we see that its success rate surpasses the *VisualRHP*$^{3\times3}_{cas}$. A possible explanation is the conservative policy learned by *CartesianRHP*$^{3\times3}_{cas}$ makes it more robust against noise on the shape of the objects. Because of the engineered feature vector of the state representation, the shape of the objects are unknowable to the policy. Hence, each of the DNNs in *CartesianRHP*$^{3\times3}_{cas}$, one for every specific number of objects, has converged to a behavior that increases the chances of success irrespective of the shapes but at the expense of a lower action efficiency. In contrast, *VisualRHP*$^{3\times3}_{cas}$ tailors the behavior to the exact shape of the objects making it more action efficient and generalizable, but also it is slightly more susceptible to high noise. We expect that a decently parameterized physics simulator would be good enough for *Visu-*

[6] The difference in the inference time between different DNN architectures is minimal and has no measurable effect on the average execution time.

[7] Standard deviation on the physics and geometric parameters with corresponding noise level: low = $0.01 \times$ *mean*, medium = $0.03 \times$ *mean*, high = $0.05 \times$ *mean*.

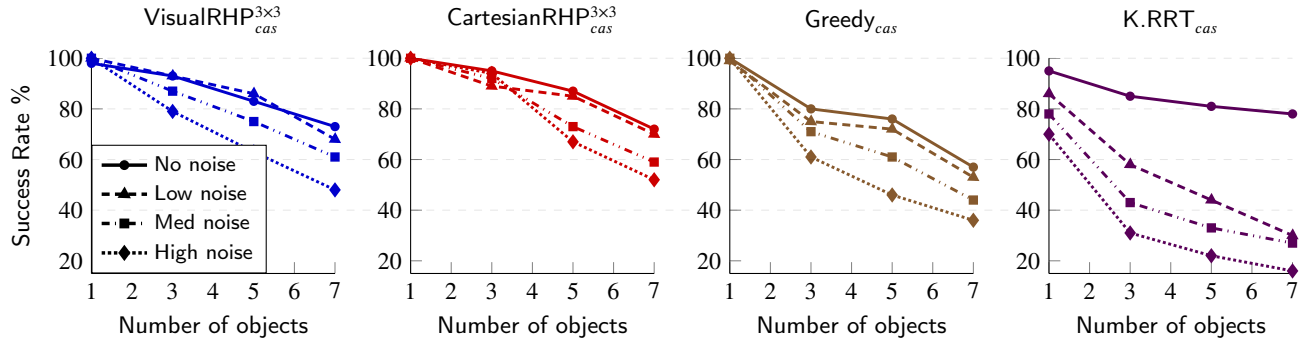


Fig. 10. Performance w. r. t. clutter density in *cas* for different noise level on the physics and geometry parameters.

alRHP$_{cas}$ to achieve high success rate.

9. Real-World experiments and Discussion

We build on the simulation results to compare the approaches evaluated in the previous section in real-world experiments that require transferable manipulation skills over different task setups. We used a *Robotiq 2F-85* two finger gripper[8] mounted on a 6-DOF *UR5* robot[9]. The robot operates over a $50cm \times 50cm$ surface and target region of $7cm$ in radius. The manipulation objects include bottles, apples, oranges, and cups. Using a top mounted RGB camera, instance segmentation is performed using a *Mask R-CNN* [19] vision system trained on the *COCO Dataset* [36]. A video of the experiments is available on https://youtu.be/raKHTnJLikQ.

Starting from a similar initial environment setup, Fig. 11 and Fig. 12 shows *K.RRT* (top), *Greedy*$_{cas}$ (middle), and *VisualRHP*$^{3\times3}_{cas}$ (bottom) tasked with manipulating an orange fruit to the target region in low and high clutter environments, respectively. *K.RRT* succeeds in solving the task in a low clutter environment. This is because problems associated with the physics discrepancy between the simulator model and the real world are mitigated by minimal physical interactions in a low clutter environment. However, in a high clutter environment, small discrepancies between the physics model of the simulator and the real world, e. g., gripper-orange-bottle interactions, are compounded causing task failure as shown by the red apple falling outside the surface edge. Further, *Greedy*$_{cas}$ also performs well in a low clutter environment. It can react fast to a dynamic environment. We see the robot chasing the orange after it was unintentionally knocked to the side of the table. The high clutter environment shows that *Greedy*$_{cas}$ is prone to getting trapped in some parts of the state space (cyclic or oscillatory behaviour [59]), only escaping it after executing several ineffective actions. Also, by acting solely based on the current observation the robot fails to anticipate the upcoming states as exemplified by the red apple being pushed, via the bottle, outside the surface edge. On the other hand, *VisualRHP*$^{3\times3}_{cas}$ evaluates the potential consequences of an action before being executed in the real world. At a slightly higher computation cost, the policy selects informed actions based on the predicted environment dynamics.

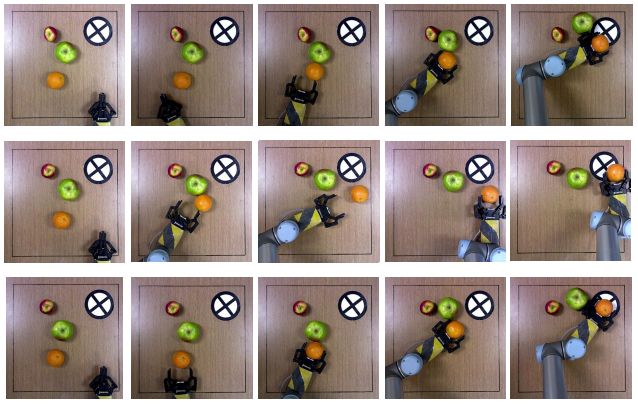


Fig. 11. *Comparing Kino-dynamic RRT* (top), *Greedy* (middle), and *VisualRHP* (bottom) policies running in continuous action space in **low** clutter environment. The "orange" is the desired object.

In Fig. 13 and with the bottle being the desired object, we observe two distinct strategies that *CartesianRHP*$^{3\times3}_{cas}$ and *VisualRHP*$^{3\times3}_{cas}$ converges to. In the top row, *CartesianRHP*$^{3\times3}_{cas}$ drives the robot around the apple obstacle. Unaware of the actual geometry of the bottle, the robot barely manages to push the bottle to the target region. In the bottom row, the robot behaviour, controlled by *VisualRHP*$^{3\times3}_{cas}$, exhibits awareness of the geometries of the objects by approaching the bottle from a graspable angle and manoeuvring it to the target region.

Additionally, Fig. 14 shows the robot being tasked with manipulating a small apple in the top row and a large apple in the bottom row using *VisualRHP*$^{3\times3}_{cas}$. When the geometry of desired object is relatively small, i. e., it fits within the fingers of the gripper, the robot manoeuvres its way through the clutter, grasps of the apple, and pulls it to the target region. Whereas, when the geometry of the desired object is relatively large, the robot resorts to pushing it towards the target region.

Fig. 15 compares the robot behavior in discrete (top) and continuous (bottom) action spaces. In this experiment the robot is tasked with manipulating the bottle to the target re-

[8] https://robotiq.com/products/2f85-140-adaptive-robot-gripper
[9] https://www.universal-robots.com/products/ur5-robot/

Fig. 12. Comparing *Kino-dynamic RRT* (top), *Greedy* (middle), and *VisualRHP* (bottom) policies running in continuous action space in **high** clutter environment. The "orange" is the desired object.

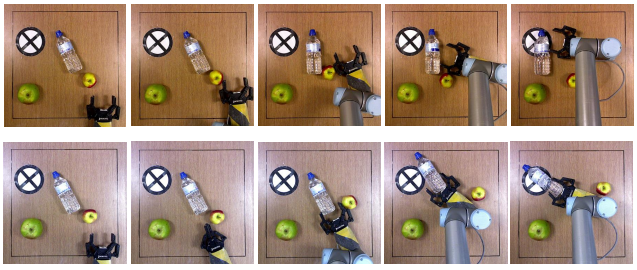


Fig. 13. Comparing *CartesianRHP* (top) and *VisualRHP* (bottom) policies running in continuous action space. The "bottle" is the desired object.

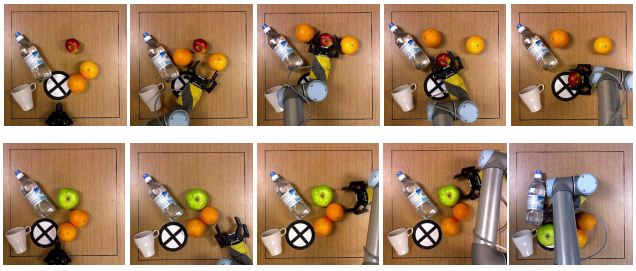


Fig. 14. Comparing *VisualRHP*, running in continuous action space, in manipulating **small** (top) and **large** (bottom) desired object. The "apple" is the desired object.

gion. Having only access to discrete actions, the robot performs several actions, particularly when the robot is positioned under the bottle, before the robot is able to get the bottle within the gripper fingers and then pushes it to the target region. In the continuous action space, the robot performs fewer actions to position itself directly under the bottle such that it can be grasped, rotated, then pushed towards the target region.

Lastly, we show $VisualRHP^{3\times3}_{cas}$ operating in challenging environment setups where the desired object is surrounded by obstacles with little room to manoeuvre. In Fig. 16, with the bottle being the desired object, the robot leverages the obstacles by pushing the larger apple which in turn is pushing

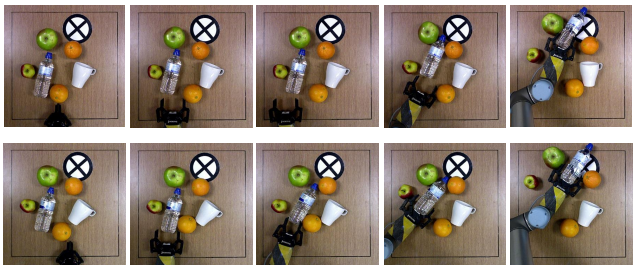


Fig. 15. Comparing *VisualRHP* in **discrete** (top) and **continuous** (bottom) action spaces. The "bottle" is the desired object.

Fig. 16. Manipulation examples with *VisualRHP* in **challenging environment setups**. The "bottle" is the desired object.

against the desired object and driving it to the target region. In Fig. 1, with the orange being the desired object, $VisualRHP^{3\times3}_{cas}$ does not only use its control over the opening and closing of the gripper for prehensile actions, but also as a tool for the robot to squeeze its way through the clutter and then opening the gripper for clearing the robot's path to the desired object.

10. Conclusions

We introduced an approach for physics-based manipulation tasks in cluttered real-world environments with limited available working space. The results show the potential of learning-based heuristic-guided RHP when presented with an expressive yet efficiently searchable state space representation. The conducted ablation study provides strong evidence for the necessity of the different components of VisualRHP. (*i*) The abstract image-based representation provides the basis for transferable and generalizable manipulation skills. The robot was shown performing in environments with different clutter densities and object shapes while also handling a variety of desired objects. (*ii*) The two learning algorithms were shown capable of learning robust heuristics to un-modelled dynamics. A learned heuristic with IL and RL drives VisualRHP to achieve high success rate and action efficiency compared to a handcrafted heuristic or open-loop execution. When using the continuous action space heuristic, the robot benefits from finer control over its actions resulting in higher action efficiency relative to operating in a discrete action space. (*iii*) The closed-loop control scheme that alternates between real-world execution and VisualRHP in a physics simulator ensures a balance between real-time execution and informed action generation for solving sequential decision making problems.

The success of our system relies on the full observability of the environment. However, full observability may not

always be available when operating in a tight work-space. A possible future line of inquiry is to incorporate partial observability into the decision making process. An approach which includes partial information in the state representation or the DNN structure may prove useful.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., et al., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/.

[2] Agboh, W.C., Dogar, M.R., 2018. Real-time online re-planning for grasping under clutter and uncertainty, in: IEEE-RAS 18th International Conference on Humanoid Robots, IEEE.

[3] Anthony, T., Tian, Z., Barber, D., 2017. Thinking fast and slow with deep learning and tree search, in: Adv Neural Inf Process Syst.

[4] Bejjani, W., 2015. Automated Planning of Whole-body Motions for Everyday Household Chores with a Humanoid Service Robot. Master's thesis. Technische Universität Dortmund.

[5] Bejjani, W., Dogar, M.R., Leonetti, M., 2019. Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6562–6569.

[6] Bejjani, W., Papallas, R., Leonetti, M., Dogar, M.R., 2018. Planning with a receding horizon for manipulation in clutter using a learned value function, in: IEEE-RAS 18th International Conference on Humanoid Robots, IEEE.

[7] Bianco, S., Cadene, R., Celona, L., Napoletano, P., 2018. Benchmark analysis of representative deep neural network architectures. IEEE Access 6, 64270–64277.

[8] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., 2012. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games 4.

[9] Catto, E., 2015. Box2d. http://box2d.org/.

[10] Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., Mouret, J.B., 2019. A survey on policy search algorithms for learning robot controllers in a handful of trials. IEEE Transactions on Robotics .

[11] Clavera, D.I., Abbeel, P., 2017. Policy transfer via modularity. IROS. IEEE .

[12] Correll, N., Bekris, K.E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J.M., Wurman, P.R., 2016. Analysis and observations from the first amazon picking challenge. IEEE Transactions on Automation Science and Engineering 15, 172–188.

[13] Dogar, M., Hsiao, K., Ciocarlie, M., Srinivasa, S., 2012. Physics-based grasp planning through clutter, in: Robotics: Science and Systems.

[14] Eppner, C., Höfer, S., Jonschkowski, R., Martín-Martín, R., Sieverling, A., Wall, V., Brock, O., 2016. Lessons from the amazon picking challenge: Four aspects of building robotic systems., in: Robotics: science and systems.

[15] Fang, K., Bai, Y., Hinterstoisser, S., Savarese, S., Kalakrishnan, M., 2018. Multi-task domain adaptation for deep learning of instance grasping from simulation, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 3516–3523.

[16] Fujimoto, S., Van Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477 .

[17] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 .

[18] Haustein, J.A., King, J., Srinivasa, S.S., Asfour, T., 2015. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states, in: ICRA, IEEE.

[19] He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask r-cnn, in: ECCV.

[20] Hernandez, C., Bharatheesha, M., Ko, W., Gaiser, H., Tan, J., van Deurzen, K., de Vries, M., Van Mil, B., van Egmond, J., Burger, R., et al., 2016. Team delft's robot winner of the amazon picking challenge 2016, in: Robot World Cup, Springer.

[21] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., et al., 2017. Learning from demonstrations for real world reinforcement learning. arXiv preprint arXiv:1704.03732 .

[22] James, S., Davison, A.J., Johns, E., 2017. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. arXiv preprint arXiv:1707.02267 .

[23] James, S., Johns, E., 2016. 3d simulation for robot arm control with deep q-learning. arXiv preprint arXiv:1609.03759 .

[24] Jeong, R., Aytar, Y., Khosid, D., Zhou, Y., Kay, J., Lampe, T., Bousmalis, K., Nori, F., 2019. Self-supervised sim-to-real adaptation for visual robotic manipulation. arXiv preprint arXiv:1910.09470 .

[25] Johnson, A.M., King, J.E., Srinivasa, S., 2016. Convergent planning. IEEE Robotics and Automation Letters , 1044–1051.

[26] Kartal, B., Hernandez-Leal, P., Taylor, M.E., 2018. Using monte carlo tree search as a demonstrator within asynchronous deep rl. arXiv preprint arXiv:1812.00045 .

[27] Kartal, B., Hernandez-Leal, P., Taylor, M.E., 2019. Action guidance with mcts for deep reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, pp. 153–159.

[28] Kimmel, A., Shome, R., Bekris, K., 2019. Anytime motion planning for prehensile manipulation in dense clutter. Advanced Robotics 33, 1175–1193.

[29] King, J.E., Haustein, J.A., Srinivasa, S.S., Asfour, T., 2015. Nonprehensile whole arm rearrangement planning on physics manifolds, in: ICRA, IEEE.

[30] Kitaev, N., Mordatch, I., Patil, S., Abbeel, P., 2015. Physics-based trajectory optimization for grasping in cluttered environments, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 3102–3109.

[31] Kloss, A., Schaal, S., Bohg, J., 2017. Combining learned and analytical models for predicting action effects. arXiv preprint arXiv:1710.04102 .

[32] Kocsis, L., Szepesvári, C., 2006. Bandit based monte-carlo planning, in: European conference on machine learning, Springer.

[33] Konietschke, R., Hirzinger, G., 2009. Inverse kinematics with closed form solutions for highly redundant robotic systems, in: 2009 IEEE International Conference on Robotics and Automation, IEEE. pp. 2945–2950.

[34] Koval, M.C., King, J.E., Pollard, N.S., Srinivasa, S.S., 2015. Robust trajectory selection for rearrangement planning as a multi-armed bandit problem, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE.

[35] Leidner, D., Bartels, G., Bejjani, W., Albu-Schäffer, A., Beetz, M., 2018. Cognition-enabled robotic wiping: Representation, planning, execution, and interpretation. Robotics and Autonomous Systems .

[36] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: Common objects in context, in: ECCV, Springer.

[37] Liu, R., Lehman, J., Molino, P., Such, F.P., Frank, E., Sergeev, A., Yosinski, J., 2018. An intriguing failing of convolutional neural networks and the coordconv solution, in: Advances in Neural Information Processing Systems, pp. 9605–9616.

[38] Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., Mordatch, I., 2018. Plan online, learn offline: Efficient learning and exploration via model-based control. arXiv preprint arXiv:1811.01848 .

[39] Mittal, M., Gallieri, M., Quaglino, A., Salehian, S.S.M., Koutník, J., 2020. Neural lyapunov model predictive control. arXiv preprint arXiv:2002.10451 .

[40] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning, in: International conference on machine learning.

[41] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 .

[42] Morrison, D., Tow, A.W., Mctaggart, M., Smith, R., Kelly-Boxall, N., Wade-Mccue, S., Erskine, J., Grinover, R., Gurman, A., Hunn, T., et al., 2018. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 7757–7764.

[43] Muhayyuddin, Moll, M., Kavraki, L., Rosell, J., 2017. Randomized Physics-based Motion Planning for Grasping in Cluttered and Uncertain Environments. ArXiv e-prints .

[44] Papallas, R., Cohn, A.G., Dogar, M.R., 2020. Online replanning with human-in-the-loop for non-prehensile manipulation in clutter—a trajectory optimization based approach. IEEE Robotics and Automation Letters .

[45] Papallas, R., Dogar, M.R., 2020. Non-prehensile manipulation in clutter with human-in-the-loop, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 6723–6729.

[46] Peng, X.B., Andrychowicz, M., Zaremba, W., Abbeel, P., 2017. Sim-to-real transfer of robotic control with dynamics randomization. arXiv preprint arXiv:1710.06537 .

[47] Ratliff, N.D., Bagnell, J.A., Zinkevich, M.A., 2006. Maximum margin planning, in: Proceedings of the 23rd international conference on Machine learning, pp. 729–736.

[48] Riccio, F., Capobianco, R., Nardi, D., 2018. Dop: Deep optimistic planning with approximate value function evaluation. arXiv preprint arXiv:1803.08501 .

[49] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., Springenberg, J.T., 2018. Learning by playing-solving sparse reward tasks from scratch. arXiv preprint arXiv:1802.10567 .

[50] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization, in: International conference on machine learning, pp. 1889–1897.

[51] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

[52] Shome, R., Tang, W.N., Song, C., Mitash, C., Kourtev, H., Yu, J., Boularias, A., Bekris, K.E., 2019. Towards robust product packing with a minimalistic end-effector, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE. pp. 9007–9013.

[53] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al., 2017. Mastering the game of go without human knowledge. Nature 550.

[54] Song, H., Haustein, J.A., Yuan, W., Hang, K., Wang, M.Y., Kragic, D., Stork, J.A., 2019. Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting. arXiv preprint arXiv:1912.07024 .

[55] Thananjeyan, B., Balakrishna, A., Rosolia, U., Li, F., McAllister, R., Gonzalez, J.E., Levine, S., Borrelli, F., Goldberg, K., 2020. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. IEEE Robotics and Automation Letters .

[56] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P., 2017. Domain randomization for transferring deep neural networks from simulation to the real world, in: IROS, IEEE.

[57] Tong, X., Liu, W., Li, B., 2019. Enhancing rolling horizon evolution with policy and value networks, in: 2019 IEEE Conference on Games (CoG), IEEE. pp. 1–8.

[58] Viereck, U., Pas, A.t., Saenko, K., Platt, R., 2017. Learning a visuomotor controller for real world robotic grasping using simulated depth images. arXiv preprint arXiv:1706.04652 .

[59] Wagner, P., 2011. A reinterpretation of the policy oscillation phenomenon in approximate policy iteration, in: Advances in Neural Information Processing Systems, pp. 2573–2581.

[60] Yuan, W., Hang, K., Kragic, D., Wang, M.Y., Stork, J.A., 2019. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. Robotics and Autonomous Systems .

[61] Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., Funkhouser, T., 2018a. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. arXiv preprint arXiv:1803.09956 .

[62] Zeng, A., Song, S., Yu, K.T., Donlon, E., Hogan, F.R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., et al., 2018b. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 1–8.