Visual Foresight Tree for Object Retrieval from Clutter with Nonprehensile Rearrangement

Baichuan Huang, Shuai D. Han, Jingjin Yu, and Abdeslam Boularias

***Abstract*—This paper considers the problem of retrieving an object from many tightly packed objects using a combination of robotic pushing and grasping actions. Object retrieval in dense clutter is an important skill for robots to operate in households and everyday environments effectively. The proposed solution, Visual Foresight Tree (VFT), intelligently rearranges the clutter surrounding a target object so that it can be grasped easily. Rearrangement with nested nonprehensile actions is challenging as it requires predicting complex object interactions in a combinatorially large configuration space of multiple objects. We first show that a deep neural network can be trained to accurately predict the poses of the packed objects when the robot pushes one of them. The predictive network provides visual foresight and is used in a tree search as a state transition function in the space of scene images. The tree search returns a sequence of consecutive push actions yielding the best arrangement of the clutter for grasping the target object. Experiments in simulation and using a real robot and objects show that the proposed approach outperforms model-free techniques as well as model-based myopic methods both in terms of success rates and the number of executed actions, on several challenging tasks.**

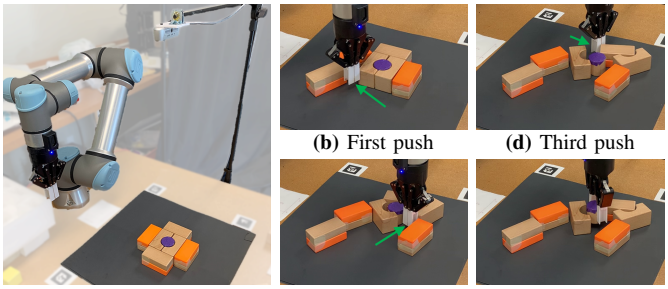
A video introducing VFT, with robot experiments, is accessible at https://youtu.be/7cL-hmgvyec. The full source code is available at https://github.com/arc-l/vft.

***Index Terms*—Deep Learning in Grasping and Manipulation, Learning from Experience, Visual Learning**

I. Introduction

In many application domains, robots are tasked with retrieving objects that are surrounded by multiple tightly packed objects. To enable the grasping of target object(s), a robot needs to re-arrange the scene to create sufficient clearance before attempting a grasp. Scene rearrangement can be achieved through nested sequential push actions, each moving multiple objects simultaneously. In this paper, we address the problem of finding the minimum number of push actions to create a scene where the target object can be grasped and retrieved.

To solve the object retrieval problem, the robot must imagine how the scene would look like after any given sequence of pushing actions, and select the shortest sequence that leads to a state where the target object can be grasped. The huge combinatorial search space makes this problem computationally challenging, hence the need for efficient planning algorithms, as well as fast predictive models that can return the predicted future states in a few milliseconds. Moreover, objects in clutter typically have unknown physical properties such as mass and friction coefficients. While it is possible to utilize off-the-shelf physics engines to simulate contacts and collisions of rigid objects in clutter, simulation is highly sensitive to the accuracy of the provided mechanical parameters. To overcome the problem of manually specifying these parameters, and to enable full autonomy of the robot, most recent works on object manipulation utilize machine learning techniques to train predictive models from data [1]–[3]. The predictive models take the state of the robot's environment a control action as inputs and predict the state after applying the control action.



(a) Hardware setup **(b)** First push **(c)** Second push **(d)** Third push **(e)** Grasp

Fig. 1: (a) The hardware setup for object retrieval in a clutter includes a Universal Robots UR-5e manipulator with a Robotiq 2F-85 two-finger gripper, and an Intel RealSense D435 RGB-D camera. The objects are placed in a square workspace. (b)(c)(d) Three push actions (shown with green arrows) are used to create space accessing the target (purple) object. The push directions are toward top-left, top-right, and bottom-right, respectively. (e) The target object is successfully grasped and retrieved.

In this work, we propose to employ *visual foresight trees* (VFT) to address the computational and modeling challenges related to the object retrieval problem. A key building block of VFT is a Convolutional Neural Networks (CNN) extending DIPN [4], capable of predicting multi-step push outcomes involving multiple objects. A second CNN evaluates the graspability of the target object in predicted future images. A Monte Carlo Tree Search utilizes the two CNNs to obtain the shortest sequence of pushing actions that lead to an arrangement where the target can be grasped.

To our knowledge, the proposed technique is the first model-based learning solution to the object retrieval problem. Extensive experiments on the real robot and objects are shown in Fig. 1 demonstrate that the proposed approach succeeds in retrieving target objects with manipulation sequences that are shorter than model-free reinforcement learning techniques and a limited-horizon planning technique.

Manuscript received: May, 5, 2021; Revised July, 19, 2021; Accepted October, 14, 2021.

This paper was recommended for publication by Editor Markus Vincze upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF awards IIS-1845888, IIS-1734492, IIS-1846043, CCF-1934924, IIS-1734492, IIS-1846043, and IIS-2132972.

B. Huang, S. D. Han, J. Yu, and A. Boularias are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. {baichuan.huang, shuai.han, jingjin.yu, abdeslam.boularias}@rutgers.edu

Digital Object Identifier (DOI): see top of this page.

II. Related works

Grasping. Robotic grasping methods are generally categorized in two main categories: *analytical* and *data-driven* [5]. Analytical approaches rely on precise 3D and mechanical models of objects to simulate *force-closure* or *form-closure* grasps [6]–[8]. Since material properties, such as mass and friction coefficients, are generally difficult to measure, most recent techniques have shifted toward learning directly grasp success probabilities from data. Most data-driven methods focused on isolated objects [9]–[12]. Learning to grasp in cluttered scenes was explored in recent works [13]–[15]. More recent learning techniques were adapted to grasp objects clutter. For example, a hierarchy of supervisors was used for learning to grasp objects in clutter from demonstrations [16]. CNNs, such as Dex-net 4.0 [17], were trained to detect grasp 6D poses in point clouds [18]. A composition of a suction cup and a gripper in [19] was shown to produce more stable grasps learned with CNNs. A randomized physics-based motion planning technique for grasping in cluttered and uncertain environments was also presented in [3]. A large-scale benchmark for general object grasping was introduced in [20]. *Sim-to-real* transfer was adopted in [21] for data-efficient learning of robotic grasps.

Object Singulation. A closely related problem is the singulation of individual items [22], i.e., isolating an item to facilitate its retrieval, typically achieved through a combination of pushing and grasping actions. In contrast with the present work, pushing in singulation is typically performed *model-free* by using reactive policies without explicitly reasoning about future states [23]. Singulation does not generally require long-horizon reasoning. For instance, linear push policies were learned in [24] to increase grasp access for robot bin picking, by using model-free reinforcement learning. The tasks considered in [24] can be solved through single pushing actions because of the lower density of clutter compared to the tasks considered in our work. While the focus of the present work is on sequential pushing actions, a new 6-DOF grasping method [25] was devised to create clearance for an object by picking and placing obstacles away. The 6-DOF grasping was also combined with a push policy [26]. Results reported in [4], [26] show, however, this combination [26] is less efficient on the same tasks than the one-step reasoning method [4] that serves as one of the baselines in our experiments.

Rearrangement Planning. Object retrieval in clutter is closely related to rearrangement planning. The approach recently presented in [27] also uses a Monte Carlo tree search, but the objectives of rearrangement tasks are different from ours. In object retrieval, we focus on finding the minimum number of pre-grasp pushing actions that lead to grasping a single target object. This objective requires highly accurate predictions of future poses of individual objects in clutter.

Object Retrieval. Several other works also addressed the problem of retrieving a target object from clutter. Some of these works focus on online planning for object search under partial observability without learning [28]. Other related works learn only the quality of pushing and grasping actions [29], without visual foresight, which is necessary for tightly packed clutter. Similarly, scene exploration and object search were learned using model-free reinforcement learning, based on active and interactive perception [30], and teacher-aided exploration [31]. A planning approach with a human operator guiding a robot to reach for a target object in clutter was presented in [32]. In contrast to these approaches, ours is fully autonomous. The work presented in [33] is most related to ours, with a similar robotic setup and objects. However, it is based on deep Q-learning, which is model-free and does not predict future states. We show in Section VI that our model-based technique significantly outperforms the one from [33] on the same tasks considered in [33] as well as on more challenging ones.

III. Preliminaries

A. Problem Statement

The Object Retrieval from Clutter (ORC) challenge asks a robot manipulator to retrieve a target object from a set of objects densely packed together. The objects may have different shapes, sizes, and colors. Objects other than the target object are unknown a priori. Focusing on a mostly planar setup, the following assumptions are made: 1) The hardware setup (Fig. 1a) contains a manipulator, a planar workspace with a uniform background color, and a camera on top of the workspace. 2) The objects are rigid and are amenable to the gripper's prehensile and non-prehensile capabilities, limited to straight-line planar push actions and top-down grasp actions. 3) The objects are confined to the workspace without overlapping. As a result, the objects are visible to the camera. 4) The target object, to be retrieved, is visually distinguishable from the others. Under these assumptions, the *objective* is to retrieve only the target object, while minimizing the number of pushing/grasping actions that are used. Each grasp or push is considered as one atomic action. While a mostly planar setup is assumed in our experiments, the proposed data-driven solution is general and can be applied to arbitrary object shapes and arrangements. In the experiments, we mainly work with woodblocks; we also evaluate the proposed approach on novel objects such as soapboxes, which are challenging as their widths are close to the maximum distances between the gripper's fingers.

B. Manipulation Motion Primitives

Similar to studies closely related to the ORC challenge, e.g., [4], [33], [34], we employ a set of pre-defined and parameterized pushing/grasping manipulation primitives. The decision-making problem then entails the search for the optimal order and parameters of these primitives. A grasp action $a^{\text{grasp}} = (x, y, \theta)$ is defined as a top-down overhead grasp motion at image pixel location (x, y), with the end-effector rotated along with the world z-axis by θ degrees. In our implementation, a grasp center (x, y) can be any pixel in a down-sampled 224×224 image of the planar scene, while rotation angle θ can be one of 16 values evenly distributed between 0 and 2π. To perform a complete grasp action, the manipulator moves the open gripper above the specified location, then moves the gripper downwards until a contact with the target object is detected, closes the fingers, and transfers the grasped object outside of the workspace.

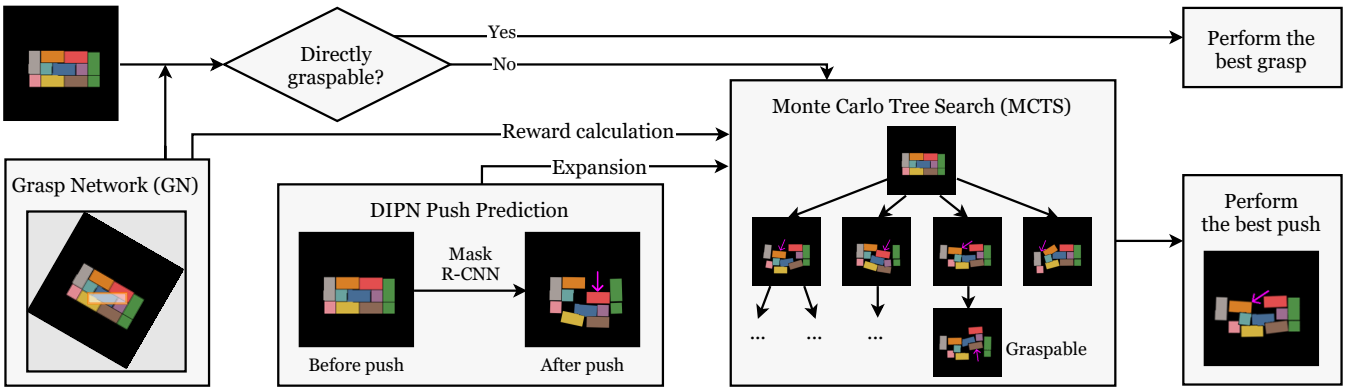


Fig. 2: Overview of the proposed technique for object retrieval from clutter with nonprehensile rearrangement. The problem is iteratively solved by observing the environment at each time step, taking the current state as input, and returning the best action. It is repeated until the object is retrieved.

When objects are densely packed, the target object is generally not directly graspable due to collisions between the gripper and surrounding objects. When this happens, non-prehensile push actions can be used to create opportunities for grasping. For a push action $a^{\text{push}} = (x_0, y_0, x_1, y_1)$, the gripper performs a quasi-static horizontal motion. Here, (x_0, y_0) and (x_1, y_1) are the start and end location of the gripper center, respectively. The gripper's orientation is fixed along the motion direction during a push maneuver.

IV. Overview of the Proposed Approach

When objects are tightly packed, the robot needs to carefully select an appropriate sequence of pushes that create a sufficient volume of empty space around the target object before attempting to grasp it. In this work, we are interested in challenging scenarios where multiple push actions may be necessary to de-clutter the surroundings of the target, and where the location, direction, and duration of each push action should be carefully optimized to minimize the total number of actions. Collisions among multiple objects often occur while pushing a single object, further complicating the matter. To address the challenge, we propose a solution that uses a neural network to forecast the outcome of a sequence of push actions in the future, and estimates the probability of succeeding in grasping the target object in the resulting scene. The optimal push sequence is selected based on the forecasts.

A high-level description of the proposed solution pipeline is depicted in Fig. 2. At the start of a planning iteration, an RGB-D image of the scene is taken, and the objects are detected and classified as *unknown clutter* or *target object*. With the target object located, a second network called Grasp Network (GN) predicts the probability of grasping the target. GN is a Deep Q-Network (DQN) [35] adopted from prior works [4], [34] for ORC. It takes the image input, and outputs the estimated grasp success probability for each grasp action. The target object is considered directly graspable if the maximum estimated grasp success probability is larger than a threshold. The robot executes the corresponding optimal grasp action; otherwise, push actions must be performed to create space for grasping.

When push actions are needed, the next action is selected using Monte-Carlo Tree Search (MCTS). In our implementation, which we call the Visual Foresight Tree (VFT), each search state corresponds to an image observation of the workspace. Given a push action and a state, VFT uses the Deep Interaction Prediction Network (DIPN) [4] as the state transition function. Here, DIPN is a network that predicts the motions of multiple objects and generates a synthetic image corresponding to the scene after the imagined push. VFT uses GN to obtain a reward value for each search node and detect whether the search terminates. Both DIPN and GN are trained offline on different objects.

V. Visual Foresight Tree

This section discusses the three main components of VFT: GN, DIPN, and Monte-Carlo Tree Search (MCTS).

A. Grasp Network

The Grasp Network (GN), adapted from [4], takes the image s_t as input, and outputs a pixel-wise reward prediction $R(s_t) = [R(s_t, a^1), \ldots, R(s_t, a^n)]$ for grasps $a^1, \ldots, a^n$. The output is a 2D map with the same size as the input image, and where each point contains the predicted reward of performing a grasp at the corresponding input pixel. Table $R(s_t)$ is a one channel image with the same size as input image s_t (224×224 in our experiments), and a value $R(s_t, a^i)$ represents the expected reward of the grasp at the corresponding action. To train GN, we set the reward to be 1 for grasps where the robot successfully picks up only the target object, and 0 otherwise. GN is the reward estimator for states in VFT (in Section V-C).

A grasp action $a^{\text{grasp}} = (x, y, \theta)$ specifies the grasp location and the end-effector angle. GN is trained while keeping the orientation of the end-effector fixed relative to the support surface, while randomly varying the poses of the objects. Therefore, GN assumes that the grasps are aligned to the principal axis of the input image. To compute reward R for grasps with $\theta \neq 0$, the input image is rotated by θ before passing it to GN. As a result, for each input image, GN generates 16 different grasp R reward tables.

The training process of the GN used in this work is based on previous works [4], [34] but differs in terms of objectives, which requires a significant modification, explained in the following. The objective in previous works is to grasp all the objects; the goal of ORC is to retrieve a specific target among a large number of obstacles. We noticed from our experiments that if GN is trained to grasp all the objects, then a greedy policy will be learned, and it will always select the most accessible object to grasp. In contrast, all other objects that

can also be directly grasped are ignored because they have low predicted rewards. This causes the problem that GN cannot correctly predict the grasp success rate of a specific target object. One straightforward adaption to this new objective is only to give reward when the grasp center is inside the target object, which is the approach that was followed in [33]. However, we found that we can achieve a higher sample efficiency by providing a reward for successfully grasping any object. The proposed training approach is similar in spirit to Hindsight Experience Replay (HER) [36]. To balance between exploration and exploitation, grasp actions are randomly sampled from $P(s, a^{\text{grasp}}) \propto bR(s, a^{\text{grasp}})^{b-1}$ where b is set to $3/2$ in the experiments.

After training, GN can be used for selecting grasping actions in new scenes. Since the network returns reward R for all possible grasps, and not only for the target object, the first post-processing step consists in selecting a small set of grasps that overlap with the target object. This is achieved by computing the overlap between the surface of the target object and the projected footprint of the robotic hand, and keeping only grasps that maximize the overlap. Then, grasps with the highest predicted values obtained from the trained network are ranked, and the best choice without incurring collisions is selected for execution.

B. Push Prediction Network

DIPN [4] is a network that takes an RGB-D image, 2D masks of objects, center positions of objects, and a vector of the starting and endpoints of a push action. It outputs predicted translations and rotations for each passed object. The predicted poses of objects are then used to create a synthetic image. Effectively, DIPN imagines what happens to the clutter if the robot executes a certain push.

The de-cluttering tasks considered in [4] required only single-step predictions. The ORC challenge requires highly accurate predictions for multiple consecutive pushes in the future. To adapt DIPN for ORC, we fine-tuned its architecture, replacing ResNet-18 with ResNet-10 [37] while increasing the dimension of outputs from 256 to 512 to predict motions of more objects simultaneously and efficiently. The number of decoder MLP layers is also increased to six, with sizes $[768, 256, 64, 16, 3, 3]$. Other augmentations are reported in Section VI. Finally, we trained the network with $200,000$ random push actions applied on various objects. This number is higher than the $1,500$ actions used in [4] as we aim for the accuracy needed for long-horizon visual foresight. Given a sequence of candidate push actions, the fine-tuned DIPN predicts complex interactions, e.g., Fig. 3.

C. Visual Foresight Tree Search (VFT)

We introduce DIPN for predicting single-step push outcome and GN for generating/rating grasps as building blocks for a multi-step procedure capable of long-horizon planning. A natural choice is Monte-Carlo Tree Search (MCTS) [38], which balances scalability and optimality. In essence, VFT fuses MCTS and DIPN to generate an optimal multi-step push prediction, as graded by GN. A search node in VFT corresponds to an input scene or one imagined by DIPN. MCTS prioritizes the most promising states when expanding

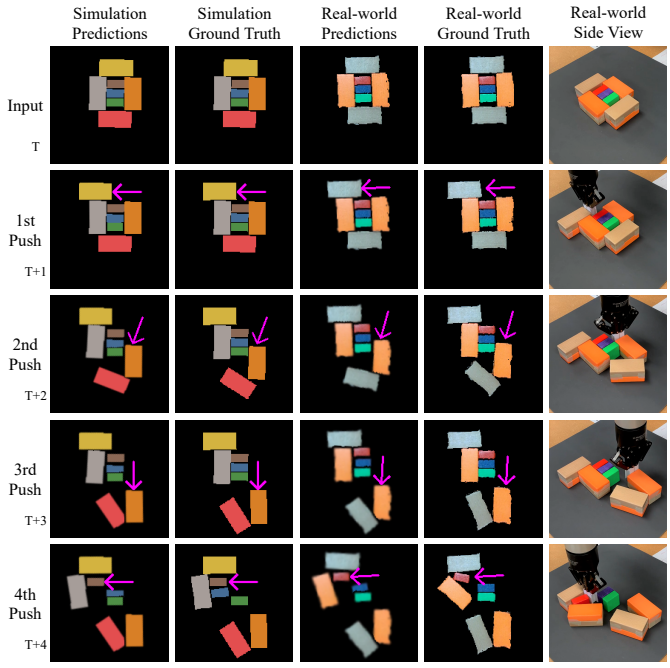


Fig. 3: Example of 4 consecutive pushes showing that DIPN can accurately predict push outcomes over a long horizon. We use purple arrows to illustrate push actions. The first and second columns are the predictions and ground truth (objects' positions after executing the pushes) in simulation. The third and fourth columns show result on a real system. The last column is the side view of the push result. Each row represents the push outcome with the previous row as the input observation.

the search tree; in VFT, such states are the ones leading to a successful target retrieval in the least number of pushes.

In a basic search iteration, MCTS has four essential steps: selection, expansion, simulation, and back-propagation. First, the *selection* stage samples a search node and a push action based on a selection function. Then, the *expansion* stage creates a child node of the selected node. After that, the reward value of the new child node is determined by a *simulation* from the node to an end state. Finally, the *back-propagation* stage updates the estimated Q-values of the parent nodes.

For describing MCTS with visual foresight, let $N(n)$ be the number of visits to a node n and $Q(n) = \{r_1, \ldots, r_{N(n)}\}$ as the estimated Q-values of each visit. We use N_{max} to denote the number of iterations the MCTS performed; we may also use an alternative computational budget to stop the search [38]. The high-level workflow of our algorithm is depicted in Alg. 1, and illustrated in Fig. 2. We will describe one iteration (line 11-29) of MCTS in VFT along with the pseudo-code in the remaining of this section.

Selection. The first step of MCTS is to select an *expandable* search node (line 12-13) using a tree policy π_{tree}. Here, *expandable* means the node has some push actions that are not tried via selection-expansion; more details of the push action space will be discussed later in the expansion part. To balance between exploration and exploitation, when the current node n_c is already fully expanded, π_{tree} uses Upper Confidence Bounds for Trees (UCT) [38] to rank its child node n_i. We customize UCT as

$$\text{UCT}(n_i, n_c) = \frac{Q^m(n_i)}{\min\{N(n_i), m\}} + C\sqrt{\frac{\ln N(n_c)}{N(n_i)}}. \quad (1)$$

Algorithm 1: Visual Foresight Tree Search

1 **Function** VFT(s_t)
2 **while** there is a target object in workspace **do**
3 $R(s_t) \leftarrow \text{GN}(s_t)$
4 **if** $\max_{a^{\text{grasp}}} R(s_t, a^{\text{grasp}}) > R^*_g$ **then**
5 Execute $\arg\max_{a^{\text{grasp}}} R(s_t, a^{\text{grasp}})$ // Grasp
6 **else** Execute MCTS(s_t) // Push
7 **Function** MCTS(s_t)**:**
8 Create root node n_0 with state s_t
9 $N(\cdot) \leftarrow 0$, $Q(\cdot) \leftarrow \varnothing$ // Default N, Q for a search node
10 **for** $i \leftarrow 1, 2, \ldots, N_{max}$ **do**
11 $n_c \leftarrow n_0$
▷ **Selection and Expansion**
12 **while** n_c is not expandable **do**
13 $n_c \leftarrow \pi_{\text{tree}}(n_c)$ // Use (1) to find a child node
14 $a^{\text{push}} \leftarrow$ sample from untried push actions in n_c
15 $n_c \leftarrow \text{DIPN}(n_c, a^{\text{push}})$ // Generate node by push prediction
▷ **Simulation**
16 $r \leftarrow 0$, $d \leftarrow 1$, $s \leftarrow n_c$.state // s is the state of n_c
17 **while** s is not a terminal state **do**
18 $a^{\text{push}} \leftarrow$ randomly select a push action in s
19 $s \leftarrow \text{DIPN}(s, a^{\text{push}})$ // Simulate to next state
20 $R(s) \leftarrow \text{GN}(s)$
21 $r \leftarrow \max\{r, \gamma^d \max_{a^{\text{grasp}}} R(s, a^{\text{grasp}})\}$
22 $d \leftarrow d + 1$
▷ **Back-propagation**
23 **while** n_c is not root **do**
24 $N(n_c) \leftarrow N(n_c) + 1$
25 $R(n_c.\text{state}) \leftarrow \text{GN}(n_c.\text{state})$
26 $r \leftarrow \max\{r, \max_{a^{\text{grasp}}} R(n_c.\text{state}, a^{\text{grasp}})\}$
27 $Q(n_c) \leftarrow Q(n_c) \cup \{r\}$ // Record the reward
28 $r \leftarrow r \cdot \gamma$
29 $n_c \leftarrow$ parent of n_c
30 $n_{\text{best}} \leftarrow \arg\max_{n_i \in \text{children of } n_0} (\text{UCT}(n_i, n_0))$
31 **return** push action a^{push} that leads to n_{best} from the root

Here, C is an exploration weight. In the first term of (1), unlike typical UCT that favours the child node that maximizes $Q(n_i)$, we keep only the most promising rollouts of n_i and denote by $Q^m(n_i)$ the average returns of the top m rollouts of n_i. In our implementation, $m = 3$ and $C = 2$. We also use (1) with parameters $m = 1$ and $C = 0$ to find the best node, and thus the best push action to execute, after the search is completed, as shown in line 30.

Expansion. Given a selected node n, we use DIPN to generate a child node by randomly choosing an untried push action a^{push} (line 14-15). The action a^{push} is uniformly sampled at random from the selected node's action space, which contains two types of push actions: 1) For each object, we apply principal component analysis to compute its feature axis. For example, for a rectangle object, the feature axis will be parallel to its long side. Four push actions are then sampled with directions perpendicular or parallel to the feature axis, pushing the object from the outside to its center. 2) To build a more complete action space, eight additional actions are evenly distributed on each object's contour, with push direction also towards the object's center.

Simulation. After we generated a new node via expansion, in line 16-22, we estimate the node's Q-value by uniformly randomly select push actions at random (line 18) and use DIPN to predict future states (line 19) until one of the following two termination criteria is met: 1) The total number of push actions used to reach a simulated state is larger than a constant D^*. 2) The maximum predicted reward value of a simulated state exceeds a threshold R^*_{gp}. In line 21, when calculating r, a discount factor γ is used to penalize a long sequence of action. Here, we use $\max$ GN to reference the maximum value in a grasp reward table. In our implementation, GN is only called once for each unique state and the output is saved by a hashmap.

Back-propagation. After simulation, the terminal grasp reward is back-propagated (line 23-29) through its parent nodes to update their $N(n)$ and $Q(n)$. Denote by r_0 the max grasp reward of a newly expanded node n_0, and $n_1, n_2, \ldots, n_k$ as the sequence of n_0's parents in the ascending order up to node n_k. With $Q(n_0) = \{r_0\}$, the Q-value of n_k in this iteration is then $\max_{0 \le j < k} \gamma^{k-j} \max Q(n_j)$, which corresponds to the max reward of states along the path [27]. Here, γ is a discount factor to penalize a long sequence of actions. As a result, for each parent n_k, $N(n_k)$ increases by 1, and $\max_{0 \le j < k} \gamma^{k-j} \max Q(n_j)$ is added to $Q(n_k)$.

VI. Experimental Evaluation

We performed an extensive evaluation of the proposed method, VFT, in simulation and on the real hardware system illustrated in Fig. 1. VFT is compared with multiple state-of-the-art approaches [4], [33], [34], with necessary modifications for solving ORC, i.e., minimizing the number of actions in retrieving a target. The results convincingly demonstrate VFT to be robust and more efficient than the compared approaches.

Both training and inference are performed on a machine with an Nvidia GeForce RTX 2080 Ti graphics card, an Intel i7-9700K CPU, and 32GB of memory.

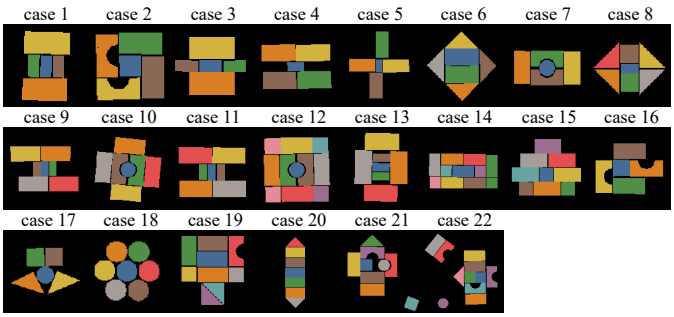


Fig. 4: 22 Test cases used in both simulation and real world experiments. The target objects are blue. Images are zoomed in for better visualization.

A. Experiment Setup

The complete test case set includes 1) the full set of 14 test cases from [33], and 2) 18 hand-designed and more challenging test cases where the objects are tightly packed. All test cases are constructed using wood blocks with different shapes, colors, and sizes. We set the workspace's dimensions to 44.8cm $\times$ 44.8cm. The size of the images is 224×224. Push actions have a minimum 5cm *effective push distance*, defined as the end-effector's moving distance after object contact. Multiple planned push actions may be concatenated if they are in the same direction and each action's end location is the same as the next action's start location. In all scenes, the target object is roughly at the center of the scene.

The hyperparameters for VFT are set as follows. The number of iterations $N_{max} = 150$. The discount factor $\gamma = 0.8$. The

maximum depth D^* of the tree is capped at 4. The terminal threshold of grasp reward $R^*_{gp} = 1.0$. Threshold R^*_g that decides to grasp or to push is 0.8 in the simulation experiments and 0.7 in the real hardware experiments. Such thresholds can potentially be fully optimized for a production system; it is not carried out in this work as reasonably good values are easily obtained while it is prohibitively time-consuming to carry out a full-scale optimization.

B. Network Training Process

VFT contains two deep neural networks: GN and DIPN. Both are trained in simulation with the same objects as used in real experiments to capture the physical properties and dynamics of the environment. No prior knowledge is given to the networks except the dimensions of the gripper fingers.

GN is trained on-policy with 20,000 grasp actions. Similar to [4], [33], [34], randomly-shaped objects are uniformly dropped onto the workspace to construct the training scenarios. A successful grasp is decided by checking the distance between grippers, which should be greater than 0. A Huber loss on the pixel where the robot performed the grasp action is used. All other pixels do not contribute to the loss during back-propagation. Image-based pre-training [4], [39] was employed to initialize the training parameters. We then train the GN by stochastic gradient descent with the momentum of 0.9, weight decay of 10^{-4}, and batch size of 12. The learning rate is set to 5×10^{-5} and by half every 2000 iteration.

DIPN [4] is trained in a supervised manner with 200,000 random push actions from simulation. In the push data set, 20% of the scenes contain randomly placed objects, and 80% contain densely packed objects. The push distance for DIPN is fixed to 7.4 cm (effective touch distance is 5 cm). In the original DIPN paper [4], the distance was 5 cm and 10 cm without considering the effective range.

We note that a total of 2000 actions (500 grasps and 1500 pushes) are sufficient for the networks to achieve fairly accurate results (see, e.g., [4]). Because training samples are readily available from simulation, it is not necessary to skimp on training data. We thus opted to train with more data to evaluate the full potential of VFT.

A Smooth L1 Loss with beta equals to 2 is used instead of 1 [4]. We train the DIPN by stochastic gradient descent with the momentum of 0.9, weight decay of 10^{-4}, and using cosine annealing schedule [40] with learning rates of learning rate of 10^{-3} for 76 epochs, and the batch size is 128.

C. Compared Methods and Evaluation Metrics

Goal-Conditioned VPG (gc-VPG). Goal-conditioned VPG (gc-VPG) is a modified version of Visual Pushing Grasping (VPG) [34], which uses two DQNs [35] for pushing and grasping predictions. VPG by itself does not focus on specific objects; it was conditioned [33] to focus on the target object to serve as a comparison point, yielding gc-VPG.

Goal-Oriented Push-Grasping. In [33], many modifications are applied to VPG to render the resulting network more suitable for solving ORC, including adopting a three-stage training strategy and an efficient labeling method [36]. For convenience, we refer to this method as go-PGN (the authors of [33] did not provide a short name for the method).

DIPN. As an ablation baseline for evaluating the utility of employing deep tree search, we replace MCTS from VFT with a search tree of depth one. In this baseline, DIPN is used to evaluate all candidate push actions. The push action whose predicted next state has the highest grasp reward for the target object is then chosen. This is similar to how DIPN is used in [4]; we thus refer to it simply as DIPN.

In our evaluation, the main metric is the total number of push and grasp actions used to retrieve the target object. For a complete comparison to [33], [34], we also list VFT's grasp success rate, which is the ratio of successful grasps in the total number of grasps during testing. The completion rate, i.e., the chance of eventually grasping the target object, is also reported. Similar to [4], when DIPN is used, a 100% completion rate often reached.

We only collected evaluation data on DIPN and VFT. For the other two baselines, gc-VPG and go-PGN, results are directly quoted from [33] (at the time of our submission, we could not obtain the trained model or the information necessary for the reproduction of gc-VPG and go-PGN). While our hardware setup is identical to that of [33], and the poses of objects are also identical, we note that there are some small differences between the evaluation setups: 1) We use PyBullet [41] for simulation, while [33] uses CoppeliaSim [42]; the physics engine is the same (Bullet). 2) [33] uses an RD2 gripper in simulation and a Robotiq 2F-85 gripper for real experiment; all of our experiments use 2F-85. 3) [33] has a 13cm push distance, while we only use a 5cm effective distance (the distance where fingers touch the objects) 4) [33] uses extra top-sliding pushes which expand the push action set. At the same time, we confirm that these relatively minor differences do not provide our algorithm any unfair advantage.

D. Simulation Studies

Fig. 5 and Table. I show the evaluation results of all algorithms on the 10 simulation test cases from [33]. Each experiment is repeated 30 times, and the average number of actions until task completion in each experiment is reported. Our proposed method, VFT, which uses an average of 2.00 actions, significantly outperforms the compared methods. Specifically, VFT uses one push action and one grasp action to solve the majority of cases, except for one instance with a half-cylinder shaped object, which is not included during the training of the networks. Interestingly, when only one push is necessary, VFT, with its main advantage as multi-step prediction, still outperforms DIPN due to its extra simulation steps. The algorithms with push prediction performs better than gc-VPG and go-PGN in all metrics.

	Completion	Grasp Success	Number of Actions
gc-VPG [33]	89.3%	41.7%	5.78
go-PGN [33]	99.0%	90.2%	2.77
DIPN [4]	100%	100%	2.30
VFT (ours)	100%	100%	**2.00**

TABLE I: Simulation results for the 10 test cases from [33].

To probe the limit of VFT's capability, we evaluated the methods on harder cases demanding multiple pushes. The test set includes 18 manually designed instances and 4 cases from [33] (see Fig. 4). As shown in Fig. 6 and Table. II, VFT uses

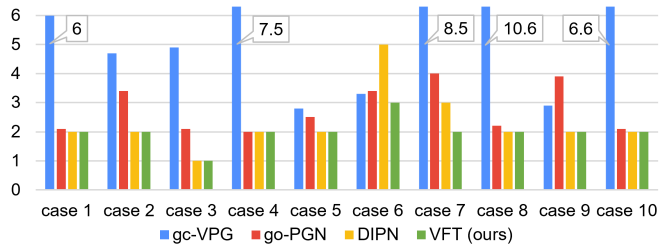


Fig. 5: Simulation results per test case for the 10 problems from [33]. The horizontal axis shows the average number of actions used to solve a problem instance: the lower, the better.

fewer actions than DIPN as VFT looks further into the future. Though we could not evaluate the performance of gc-VPG and go-PGN on these settings for direct comparison because we could not obtain the information necessary for the reproduction of these systems, notably, the average number of actions (2.45) used by VFT on harder instances is even smaller than the number of actions (2.77) go-PGN used on the 10 simpler cases.

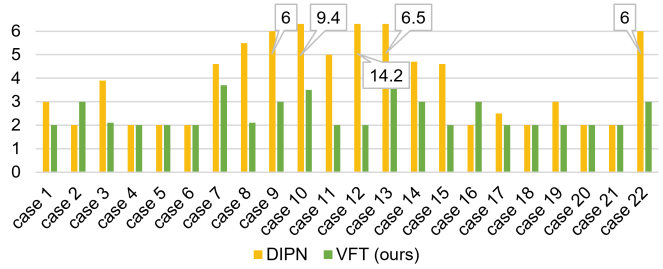


Fig. 6: Simulation result per test case for the 22 harder problems (Fig. 4). The horizontal axis shows the average number of actions used to solve a problem instance: the lower, the better.

	Completion	Grasp Success	Num. of Actions
DIPN [4]	100%	98.3%	4.31
VFT (ours)	100%	98.8%	**2.45**

TABLE II: Simulation result for the 22 test cases in Fig. 4.

E. Evaluation on a Real System

We repeated the 22 hard test cases on a real robot system (Fig. 1a). Both VFT and DIPN are evaluated. We also bring the experiment result from [33] on its 4 real test cases for comparison. All cases are repeated at least 5 times to get the mean metrics. The result, shown in Fig. 7, Table. III, and Table. IV closely matches the results from simulation. We observe a slightly lower grasp success rate due to the more noisy depth image on the real system. The real workspace's surface friction is also different from simulation. However, VFT and DIPN can still generate accurate foresight.

	Completion	Grasp Success	Num. of Actions
DIPN [4]	100%	97.0%	4.78
VFT (ours)	100%	98.5%	**2.65**

TABLE III: Real experiment results for the 22 Test cases in Fig. 4.

	Completion	Grasp Success	Num. of Actions
go-PGN [33]	95.0%	86.6%	4.62
DIPN [4]	100%	100%	4.00
VFT (ours)	100%	100%	**2.60**

TABLE IV: Real experiment results for cases 19 to 22 in Fig. 4.

We also explored our system on everyday objects (Fig. 8), where we want to retrieve a small robotic vehicle surrounded

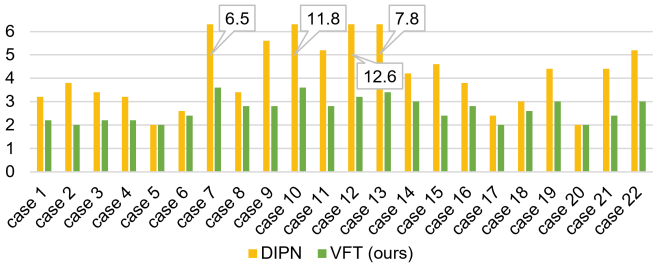


Fig. 7: Real experiment results per test case for the 22 harder problems (Fig. 4). The horizontal axis shows the average number of actions used to solve a problem instance: the lower, the better.

by soapboxes. Although the soapboxes and the small vehicles are unseen types of objects during training, the robot is able to strategically push the soapboxes away in two moves only and retrieve the vehicle.

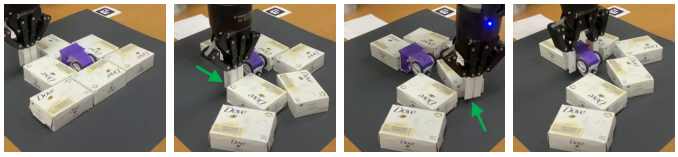


Fig. 8: Test scenario with soap boxes and masked 3D printed vehicle. Two push actions and one grasp action.

We report that the running time to decide one push action is around 3 minutes on average when the number of MCTS iterations is set to be 150. A single push prediction of DIPN took 30 milliseconds. While using the simulator as the transition function in MCTS under a similar criterion would take 8 minutes on average to decide one push action. In this letter, our primary focus is action optimization.

VII. Conclusion and Discussions

In conclusion, through an organic fusion of Deep Interaction Prediction Network (DIPN) and MCTS, the proposed Visual Foresight Tree (VFT) can make a high-quality multi-horizon prediction for optimized object retrieval from dense clutter. The effectiveness of VFT is convincingly demonstrated with extensive evaluation. As to the limitations of VFT, the time required is relatively long because of the large MCTS tree that needs to be computed. This can be improved with multi-threading because the rollouts have sufficient independence. Currently, only a single thread is used to complete the MCTS. It would also be interesting to develop a network for directly estimating the reward for rollout policy, which would reduce the inference time. This technique would be similar in spirit to the MuZero algorithm [43], which has been shown to be efficient by combining Monte Carlo tree search and learning by self-playing in an end-to-end manner. The learned rollout policy could lead to better performance. One issue related to end-to-end training is data efficiency, which is why this type of technique has been limited to games. Improving the data efficiency of end-to-end techniques is crucial to the deployment of these techniques on robotic tasks.

References

[1] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations*, 2020.

[2] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *CoRR*, vol. abs/1812.00568, 2018.

[3] Muhayyuddin, M. Moll, L. Kavraki, and J. Rosell, "Randomized physics-based motion planning for grasping in cluttered and uncertain environments," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 712–719, April 2018.

[4] B. Huang, S. D. Han, A. Boularias, and J. Yu, "Dipn: Deep interaction prediction network with application to clutter removal," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[5] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *Trans. Rob.*, vol. 30, no. 2, p. 289–309, Apr. 2014.

[6] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review." IEEE International Conference on Robotics and Automation, 2000.

[7] H. Liang, X. Ma, S. Li, M. Görner, S. Tang, B. Fang, F. Sun, and J. Zhang, "PointNetGPD: Detecting grasp configurations from point sets," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[8] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 886–900, 2012.

[9] A. Boularias, O. Kroemer, and J. Peters, "Learning robot grasping from 3-d images with markov random fields," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, 2011, pp. 1548–1553.

[10] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 2901–2910.

[11] C. Gabellieri, F. Angelini, V. Arapi, A. Palleschi, M. G. Catalano, G. Grioli, L. Pallottino, A. Bicchi, M. Bianchi, and M. Garabini, "Grasp it like a pro: Grasp of unknown objects with robotic hands based on skilled human expertise," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2808–2815, 2020.

[12] Q. Lu, M. Van der Merwe, B. Sundaralingam, and T. Hermans, "Multifingered grasp planning via inference in deep neural networks: Outperforming sampling by learning differentiable models," *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 55–65, 2020.

[13] A. Boularias, J. A. Bagnell, and A. Stentz, "Efficient optimization for autonomous robotic manipulation of natural objects," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2014, pp. 2520–2526.

[14] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," 2017.

[15] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," 2018.

[16] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg, "Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 2016, pp. 827–834.

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

[18] A. ten Pas, M. Gualtieri, K. Saenko, and R. P. Jr., "Grasp pose detection in point clouds," *CoRR*, vol. abs/1706.09911, 2017.

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

[20] H.-S. Fang, C. Wang, M. Gou, and C. Lu, "Graspnet-1billion: A large-scale benchmark for general object grasping," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 444–11 453.

[21] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," *CoRR*, vol. abs/1812.07252, 2018.

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

[23] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics Research*, N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, Eds. Cham: Springer International Publishing, 2020, pp. 405–419.

[24] M. Danielczuk, J. Mahler, C. Correa, and K. Goldberg, "Linear push policies to increase grasp access for robot bin picking," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 1249–1256.

[25] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox, "6-dof grasping for target-driven object manipulation in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6232–6238.

[26] B. Tang, M. Corsaro, G. Konidaris, S. Nikolaidis, and S. Tellex, "Learning collaborative pushing and grasping policies in dense clutter."

[27] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," *CoRR*, vol. abs/1912.07024, 2019.

[28] Y. Xiao, S. Katt, A. t. Pas, S. Chen, and C. Amato, "Online planning for target object search in clutter under partial observability," in *International Conference on Robotics and Automation*, 2019.

[29] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Multi-step retrieval of a target object occluded by clutter," *CoRR*, vol. abs/1903.01588, 2019.

[30] T. Novkovic, R. Pautrat, F. Furrer, M. Breyer, R. Siegwart, and J. I. Nieto, "Object finding in cluttered scenes using interactive perception," *CoRR*, vol. abs/1911.07482, 2019.

[31] A. Kurenkov, J. Taglic, R. Kulkarni, M. Dominguez-Kuhne, R. Martín-Martín, A. Garg, and S. Savarese, "Visuomotor mechanical search: Learning to retrieve target objects in clutter," in *IEEE/RSJ Int. Conference. on Intelligent Robots and Systems (IROS)*, 2020.

[32] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," *CoRR*, vol. abs/1904.03748, 2019.

[33] K. Xu, H. Yu, Q. Lai, Y. Wang, and R. Xiong, "Efficient learning of goal-oriented push-grasping synergy in clutter," *arXiv preprint arXiv:2103.05405*, 2021.

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

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

[36] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, 2017.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[39] L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T.-Y. Lin, "Learning to see before learning to act: Visual pre-training for manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7286–7293.

[40] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[41] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," , 2016–2019.

[42] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[43] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.