Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning

Andy Zeng[1,2], Shuran Song[1,2], Stefan Welker[2], Johnny Lee[2], Alberto Rodriguez[3], Thomas Funkhouser[1,2]
[1]Princeton University [2]Google [3]Massachusetts Institute of Technology
http://vpg.cs.princeton.edu

***Abstract*— Skilled robotic manipulation benefits from complex synergies between non-prehensile (*e.g.* pushing) and prehensile (*e.g.* grasping) actions: pushing can help rearrange cluttered objects to make space for arms and fingers; likewise, grasping can help displace objects to make pushing movements more precise and collision-free. In this work, we demonstrate that it is possible to discover and learn these synergies from scratch through model-free deep reinforcement learning. Our method involves training two fully convolutional networks that map from visual observations to actions: one infers the utility of pushes for a dense pixel-wise sampling of end effector orientations and locations, while the other does the same for grasping. Both networks are trained jointly in a Q-learning framework and are entirely self-supervised by trial and error, where rewards are provided from successful grasps. In this way, our policy learns pushing motions that enable future grasps, while learning grasps that can leverage past pushes. During picking experiments in both simulation and real-world scenarios, we find that our system quickly learns complex behaviors amid challenging cases of clutter, and achieves better grasping success rates and picking efficiencies than baseline alternatives after only a few hours of training. We further demonstrate that our method is capable of generalizing to novel objects. Qualitative results (videos), code, pre-trained models, and simulation environments are available at http://vpg.cs.princeton.edu**

I. Introduction

Skilled manipulation benefits from the synergies between non-prehensile (*e.g.* pushing) and prehensile (*e.g.* grasping) actions: pushing can help rearrange cluttered objects to make space for arms and fingers (see Fig. 1); likewise, grasping can help displace objects to make pushing movements more precise and collision-free.

Although considerable research has been devoted to both push and grasp planning, they have been predominantly studied in isolation. Combining pushing and grasping policies for sequential manipulation is a relatively unexplored problem. Pushing is traditionally studied for the task of precisely controlling the pose of an object. However, in many of the synergies between pushing and grasping, pushing plays a loosely defined role, *e.g.* separating two objects, making space in a particular area, or breaking up a cluster of objects. These goals are difficult to define or reward for model-based [1], [2], [3] or data-driven [4], [5], [6] approaches.

Many recent successful approaches to learning grasping policies, maximize affordance metrics learned from experience [7], [8] or induced by grasp stability metrics [9], [10]. However, it remains unclear how to plan sequences of actions that combine grasps and pushes, each learned in isolation. While hard-coded heuristics for supervising push-grasping policies have been successfully developed by exploiting domain-specific knowledge [11], they limit the types of synergistic behaviors between pushing and grasping that can be performed.

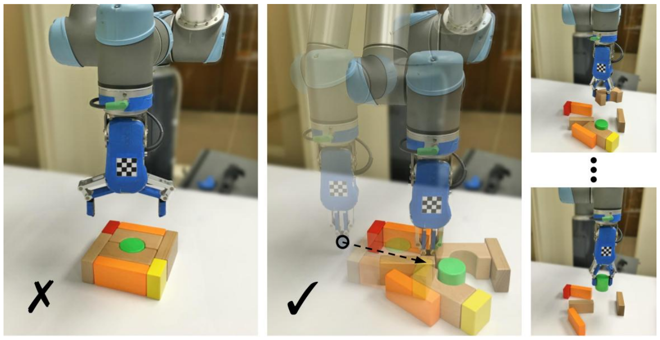

Fig. 1. **Example configuration of tightly packed blocks** reflecting the kind of clutter that commonly appears in real-world scenarios (*e.g.* with stacks of books, boxes, etc.), which remains challenging for grasp-only manipulation policies. Our model-free system is able to plan pushing motions that can isolate these objects from each other, making them easier to grasp; improving the overall stability and efficiency of picking.

In this work, we propose to discover and learn synergies between pushing and grasping from experience through model-free deep reinforcement learning (in particular, Q-learning). The key aspects of our system are:

- We learn joint pushing and grasping policies through self-supervised trial and error. Pushing actions are useful only if, in time, enable grasping. This is in contrast to prior approaches that define heuristics or hard-coded objectives for pushing motions.
- We train our policies end-to-end with a deep network that takes in visual observations and outputs expected return (*i.e.* in the form of Q values) for potential pushing and grasping actions. The joint policy then chooses the action with the highest Q value – *i.e.* , the one that maximizes the expected success of current/future grasps. This is in contrast to explicitly perceiving individual objects and planning actions on them based on hand-designed features [12].

This formulation enables our system to execute complex sequential manipulations (with pushing and grasping) of objects in unstructured picking scenarios and generalizes to

The authors would like to thank NSF (VEC 1539014/1539099), Google, Intel, NVIDIA, ABB Robotics, and Mathworks for hardware, technical, and financial support.

novel objects (unseen in training).

Training deep end-to-end policies (*e.g.* from image pixels to joint torques) with reinforcement learning on physical systems can be expensive and time-consuming due to their prohibitively high sample complexity [13], [14], [15]. To make training tractable on a real robot, we simplify the action space to a set of end-effector-driven motion primitives. We formulate the task as a pixel-wise labeling problem: where each image pixel – and image orientation – corresponds to a specific robot motion primitive (pushing or grasping) executed on the 3D location of that pixel in the scene. For pushing, this location represents the starting position of the pushing motion; for grasping, the middle position between the two fingers during parallel-jaw grasping. We train a fully convolutional network (FCN) to take an image of the scene as input, and infer dense pixel-wise predictions of future expected reward values for all pixels – and thereby all robot motion primitives executed for all visible surfaces in the scene. This pixel-wise parameterization of robot primitive actions, which we refer to as **fully convolutional action-value functions** [8], enables us to train effective pushing and grasping policies on a single robot arm in less than a few hours of robot time.

The main contribution of this paper is a new perspective to bridging data-driven prehensile and non-prehensile manipulation. We demonstrate that it is possible to train end-to-end deep networks to capture complementary pushing and grasping policies that benefit from each other through experience. We provide several experiments and ablation studies in both simulated and real settings to evaluate the key components of our system. Our results show that the pushing policies enlarge the set of scenarios in which grasping succeeds, and that both policies working in tandem produce complex interactions with objects (beyond our expectations) that support more efficient picking (*e.g.* pushing multiple blocks at a time, separating two objects, breaking up a cluster of objects through a chain of reactions that improves grasping). We provide additional qualitative results (video recordings of our robot in action), code, pre-trained models, and simulation environments at http://vpg.cs.princeton.edu

II. Related Work

Our work lies at the intersection of robotic manipulation, computer vision, and machine learning. We briefly review the related work in these domains.

Non-prehensile manipulation. Planning non-prehensile motions, such as pushing, is a fundamental problem that dates back to the early days of robotic manipulation. The literature in this area is vast, emerging early from classical solutions that explicitly model the dynamics of pushing with frictional forces [1], [2]. While inspiring, many of these methods rely on modeling assumptions that do not hold in practice [16], [4]. For example, non-uniform friction distributions across object surfaces and the variability of friction are only some of the factors that can lead to erroneous predictions of friction-modeling pushing solutions in real-world settings. While recent methods have explored data-driven algorithms for learning the dynamics of pushing [17], [18], [19], many of these works have largely focused on the execution of stable pushes for one object at a time. Modeling the larger-scale consequences of pushing in the face of severe clutter and friction variation continues to be a complex problem; effectively using these models to discover optimal policies in real settings – even more so.

Grasping. Grasping too, has been well studied in the domain of model-based reasoning; from modeling contact forces and their resistance to external wrenches [20], [21], to characterizing grasps by their ability to constrain object mobility [22]. A common approach to deploying these methods in real systems involves pre-computing grasps from a database of known 3D object models [23], and indexing them at run-time with point cloud registration for object pose estimation [24], [25]. These methods, however, typically assume knowledge of object shapes, poses, dynamics, and contact points – information which is rarely known for novel objects in unstructured environments.

More recent data-driven methods explore the prospects of training model-agnostic deep grasping policies [26], [7], [27], [9], [10], [8] that detect grasps by exploiting learned visual features, and without explicitly using object specific knowledge (*i.e.* shape, pose, dynamics). Pinto *et al.* [27] improve the performance of these deep policies by using models pre-trained on auxiliary tasks such as poking. Zeng *et al.* [8] demonstrate that using FCNs to efficiently model these policies with affordances can drastically improve run-times. Analogous to these methods, our data-driven framework is model-agnostic, but with the addition of improving the performance of grasping by incorporating non-prehensile actions like pushing.

Pushing with grasping. Combining both non-prehensile and prehensile manipulation policies is interesting, albeit an area of research that has been much less explored. The seminal work of Dogar *et al.* [11] presents a robust planning framework for push-grasping (non-prehensile motions baked within grasping primitives) to reduce grasp uncertainty as well as an additional motion primitive – sweeping – to move around obstacles in clutter. The policies in their framework, however, remain largely handcrafted. In contrast, our method is data-driven and learned online by self-supervision.

Other methods [28], [6] explore the model-free planning of pushing motions to move objects to target positions that are more favorable for pre-designed grasping algorithms – the behaviors of which are typically handcrafted, fixed, and well-known in advance. This knowledge is primarily used to define concrete goals (*e.g.* target positions) that can aid in the design or training of pushing policies. However, trying to define similar goals for data-driven model-agnostic grasping policies (where optimal behaviors emerge from experience) become less clear, as these policies are constantly learning, changing, and adapting behaviors over time with more data.

More closely related to our work is that of Boularias *et al.* [12], which explores the use of reinforcement learning for training control policies to select among push and grasp

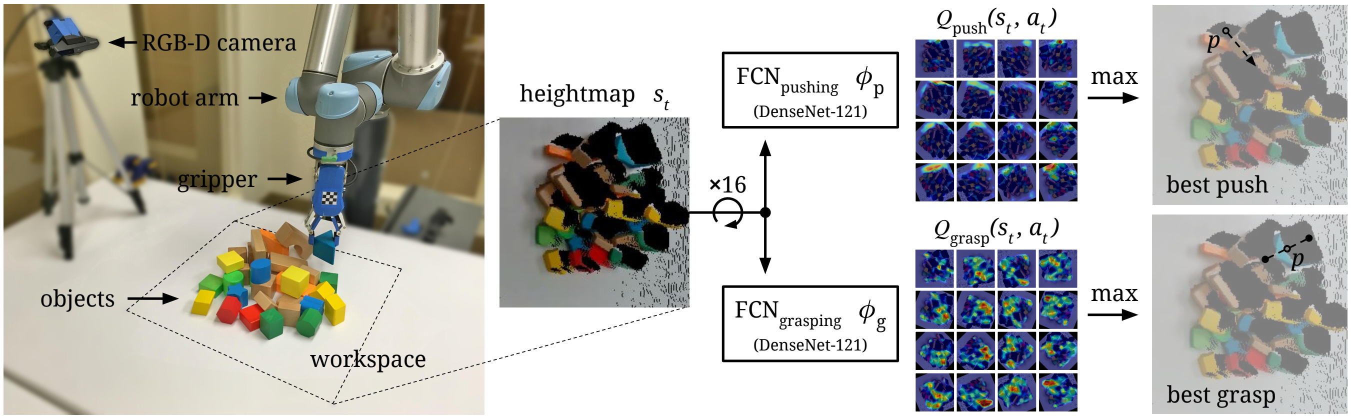
RGB-D camera
robot arm
gripper
objects
workspace
heightmap s_t
×16
FCN$_{\text{pushing}}$ ϕ_{p}
(DenseNet-121)
FCN$_{\text{grasping}}$ ϕ_{g}
(DenseNet-121)
$Q_{\text{push}}(s_t, a_t)$
$Q_{\text{grasp}}(s_t, a_t)$
max
max
p
best push
p
best grasp

Fig. 2. **Overview** of our system and Q-learning formulation. Our robot arm operates over a workspace observed by a statically mounted RGB-D camera. Visual 3D data is re-projected onto an orthographic RGB-D heightmap, which serves as a representation of the current state s_t. The heightmaps are then fed into two FCNs - one ϕ_{p} inferring pixel-wise Q values (visualized with heat maps) for pushing to the right of the heightmap and another ϕ_{g} for horizontal grasping over the heightmap. Each pixel represents a different location on which to execute the primitive. This is repeated for 16 different rotations of the heightmap to account for various pushing and grasping angles. These FCNs jointly define our deep Q function and are trained simultaneously.

proposals represented by hand-crafted features. They propose a pipeline that first segments images into objects, proposes pushing and grasping actions, extracts hand-tuned features for each action, then executes the action with highest expected reward. While inspiring, their method models perception and control policies separately (not end-to-end); it relies on model-based simulation to predict the motion of pushed objects and to infer its benefits for future grasping (those predictions are the two “features” provided to the pushing policy); it is tuned to work mainly for convex objects, and demonstrated on only one scenario with only two objects (a cylinder next to a box). In contrast, we train perception and control policies with end-to-end deep networks; we make no assumptions about the shapes or dynamics of objects (model-free), and we demonstrate that our formulation works not only for a variety of test cases with numerous objects (up to 30+), but also that it is capable of quickly generalizing to novel objects and scenarios. To the best of our knowledge, our work is the first model-free system to perform reinforcement learning of complementary pushing and grasping policies with deep networks that operate end-to-end from visual observations to actions.

III. Problem Formulation

We formulate the task of pushing-for-grasping as a Markov decision process: at any given state s_t at time t, the agent (*i.e.* robot) chooses and executes an action a_t according to a policy $\pi(s_t)$, then transitions to a new state s_{t+1} and receives an immediate corresponding reward $R_{a_t}(s_t, s_{t+1})$. The goal of our robotic reinforcement learning problem is to find an optimal policy π^* that maximizes the expected sum of future rewards, given by $R_t = \sum_{i=t}^{\infty} \gamma R_{a_i}(s_i, s_{i+1})$, *i.e.* γ-discounted sum over an infinite-horizon of future returns from time t to ∞.

In this work, we investigate the use of off-policy Q-learning to train a greedy deterministic policy $\pi(s_t)$ that chooses actions by maximizing the action-value function (*i.e.* Q-function) $Q_\pi(s_t, a_t)$, which measures the expected reward of taking action a_t in state s_t at time t. Formally, our learning objective is to iteratively minimize the temporal difference error δ_t of $Q_\pi(s_t, a_t)$ to a fixed target value y_t:

$$\delta_t = |Q(s_t, a_t) - y_t|$$

$$y_t = R_{a_t}(s_t, s_{t+1}) + \gamma\, Q(s_{t+1}, \underset{a'_t}{\operatorname{argmax}}(Q(s_{t+1}, a'_t)))$$

where a'_t is the set of all available actions.

IV. Method

This section provides details of our Q-learning formulation, network architectures, and training protocols.

A. State Representations

We model each state s_t as an RGB-D heightmap image representation of the scene at time t. To compute this heightmap, we capture RGB-D images from a fixed-mount camera, project the data onto a 3D point cloud, and orthographically back-project upwards in the gravity direction to construct a heightmap image representation with both color (RGB) and height-from-bottom (D) channels (see Fig. 2). The edges of the heightmaps are predefined with respect to the boundaries of the agent's workspace for picking. In our experiments, this area covers a 0.448^2m tabletop surface. Since our heightmaps have a pixel resolution of 224×224, each pixel spatially represents a 2^2mm vertical column of 3D space in the agent's workspace.

B. Primitive Actions

We parameterize each action a_t as a motion primitive behavior ψ (*e.g.* pushing or grasping) executed at the 3D location q projected from a pixel p of the heightmap image representation of the state s_t:

$$a = (\psi, q) \mid \psi \in \{\text{push}, \text{grasp}\}, q \twoheadrightarrow p \in s_t$$

Our motion primitive behaviors are defined as follows:

Pushing: q denotes the starting position of a 10cm push in one of $k = 16$ directions. The trajectory of the push is straight. It is physically executed in our experiments using the tip of a closed two-finger gripper.

Grasping: q denotes the middle position of a top-down parallel-jaw grasp in one of $k = 16$ orientations. During a grasp attempt, both fingers attempt to move 3cm below q (in the gravity direction) before closing the fingers. In both primitives, robot arm motion planning is automatically executed with stable, collision-free IK solves [29].

C. Learning Fully Convolutional Action-Value Functions

We extend vanilla deep Q-networks (DQN) [30] by modeling our Q-function as two feed-forward fully convolutional networks (FCNs) [31] ϕ_{p} and ϕ_{g}; one for each motion primitive behavior (pushing and grasping respectively). Each individual FCN ϕ_ψ takes as input the heightmap image representation of the state s_t and outputs a dense pixel-wise map of Q values with the same image size and resolution as that of s_t, where each individual Q value prediction at a pixel p represents the future expected reward of executing primitive ψ at 3D location q where $q \twoheadrightarrow p \in s_t$. Note that this formulation is a direct amalgamation of Q-learning with visual affordance-based manipulation [8].

Both FCNs ϕ_{p} and ϕ_{g} share the same network architecture: two parallel 121-layer DenseNet [32] pre-trained on ImageNet [33], followed by channel-wise concatenation and 2 additional 1×1 convolutional layers interleaved with nonlinear activation functions (ReLU) [34] and spatial batch normalization [35], then bilinearly upsampled. One DenseNet tower takes as input the color channels (RGB) of the heightmap, while the other takes as input the channel-wise cloned depth channel (DDD) (normalized by subtracting mean and dividing standard deviation) of the heightmap.

To simplify learning oriented motion primitives for pushing and grasping, we account for different orientations by rotating the input heightmap s_t into $k = 16$ orientations (different multiples of 22.5°) and then consider only horizontal pushes (to the right) and grasps in the rotated heightmaps. Thus, the input to each FCN ϕ_ψ is $k = 16$ rotated heightmaps, and the total output is 32 pixel-wise maps of Q values (16 for pushes in different directions, and 16 for grasps at different orientations). The action that maximizes the Q-function is the primitive and pixel with the highest Q value across all 32 pixel-wise maps: $\mathrm{argmax}_{a'_t}(Q(s_t, a'_t)) = \mathrm{argmax}_{(\psi,p)}(\phi_{\mathrm{p}}(s_t), \phi_{\mathrm{g}}(s_t))$.

Our pixel-wise parameterization of both state and action spaces enables the use of FCNs as Q-function approximators, which provides several advantages. First, the Q value prediction for each action now has an explicit notion of spatial locality with respect to other actions, as well as to the input observation of the state (*e.g.* with receptive fields). Second, FCNs are efficient for pixel-wise computations. Each forward pass of our network architecture ϕ_ψ takes on average 75ms to execute, which enables computing Q values for all 1,605,632 (*i.e.* $224 \times 224 \times 32$) possible actions within 2.5 seconds. Finally, our FCN models can converge with less training data since the parameterization of end effector locations (pixel-wise sampling) and orientations (by rotating s_t) enables convolutional features to be shared across locations and orientations (*i.e.* equivariance to translation and rotation).

Additional extensions to deep networks for Q-function estimation such as double Q-learning [36], and duelling networks [37], have the potential to improve performance but are not the focus of this work.

D. Rewards

Our reward scheme for reinforcement learning is simple. We assign $R_{\mathrm{g}}(s_t, s_{t+1}) = 1$ if a grasp is successful (computed by thresholding on the antipodal distances between gripper fingers after a grasp attempt) and $R_{\mathrm{p}}(s_t, s_{t+1}) = 0.5$ for pushes that make detectable changes to the environment (where changes are detected if the sum of differences between heightmaps exceeds some threshold τ, *i.e.* $\sum(s_{t+1} - s_t) > \tau$). Note that the intrinsic reward $R_{\mathrm{p}}(s_t, s_{t+1})$ does not explicitly consider whether a push enables future grasps. Rather, it simply encourages the system to make pushes that cause change. The synergy between pushing and grasping is learned mainly through reinforcement (see experiments in Sec. V-C).

E. Training details.

Our Q-learning FCNs are trained at each iteration i using the Huber loss function:

$$\mathcal{L}_i = \begin{cases} \frac{1}{2}(Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-})^2, \text{for } |Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}| < 1, \\ |Q^{\theta_i}(s_i, a_i) - y_i^{\theta_i^-}| - \frac{1}{2}, \text{ otherwise.} \end{cases}$$

where θ_i are the parameters of the neural network at iteration i, and the target network parameters θ_i^- are held fixed between individual updates. We pass gradients only through the single pixel p and network ϕ_ψ from which the value predictions of the executed action a_i was computed. All other pixels at iteration i backpropagate with 0 loss.

We train our FCNs ϕ_ψ by stochastic gradient descent with momentum, using fixed learning rates of 10^{-4}, momentum of 0.9, and weight decay 2^{-5}. Our models are trained in PyTorch with an NVIDIA Titan X on an Intel Xeon CPU E5-2699 v3 clocked at 2.30GHz. We train with prioritized experience replay [38] using stochastic rank-based prioritization, approximated with a power-law distribution. Our exploration strategy is ϵ-greedy, with ϵ initialized at 0.5 then annealed over training to 0.1. Our future discount γ is constant at 0.5.

In our experiments (Sec. V), we train all of our models by self-supervision with the same procedure: n objects (*i.e.* toy blocks) are randomly selected and dropped into the 0.448^2m workspace in front of the robot. The robot then automatically performs data collection by trial and error, until the workspace is void of objects, at which point n objects are again randomly dropped into the workspace. In simulation $n = 10$, while in real-world settings $n = 30$.

F. Testing details.

Since our policy is greedy deterministic during test time, it is possible for it to get stuck repeatedly executing the same action while the state representation (and thus value estimates) remain the same as no change is made to the environment. Naively weighting actions based on visit counts can also be inefficient due our pixel-wise parameterization of the action space. Hence, to alleviate this issue during testing, we prescribe a small learning rate to the network at 10^{-5} and continue backpropagating gradients through the network after each executed action. Network weights are reset before each new test run – indicated by when all objects in the workspace have been successfully grasped (*i.e.* completion) or when the number of consecutively executed actions for which there is no change to the environment exceeds 10.

V. Experiments

We executed a series of experiments to test the proposed approach, which we call **Visual Pushing for Grasping (VPG)**. The goals of the experiments are three-fold: 1) to investigate whether the addition of pushing as a motion primitive can enlarge the set of scenarios in which objects can successfully be grasped (*i.e.* does pushing help grasping), 2) to test whether it is feasible to train pushing policies with supervision mainly from the future expected success of another grasping policy trained simultaneously, and 3) to demonstrate that our formulation is capable of training effective, non-trivial pushing-for-grasping policies directly from visual observations on a real system.

A. Baseline Methods

To address these goals, we compare the picking performance of VPG to the following baseline approaches:

Reactive Grasping-only Policy (Grasping-only) is a grasping policy that uses the same pixel-wise state and action space formulation as our proposed method described in Section IV, but uses a single FCN supervised with binary classification (from trial and error) to infer pixel-wise affordance values between 0 and 1 for grasping only. This baseline is a greedy deterministic policy that follows the action which maximizes the immediate grasping affordance value at every time step t. This baseline is analogous to a self-supervised version of a state-of-the-art top-down parallel-jaw grasping algorithm [8]. For a fair comparison, we extend that method using DenseNet [32] pre-trained on ImageNet [33].

Reactive Pushing and Grasping Policy (P+G Reactive) is an augmented version of the previous baseline, but with an additional FCN to infer pixel-wise affordance values between 0 and 1 for pushing. Both networks are trained with binary classification from self-supervised trial and error, where pushing is explicitly supervised with a binary value from change detection (as described in Section IV-D). Change detection is the simplest form of direct supervision for pushing, but requires higher values of ϵ for the exploration strategy to maintain stable training. This policy follows the action which maximizes the immediate affordance value (which can come from either the pushing or grasping FCNs).

Both aforementioned baselines are reactive as they do not plan long-horizon strategies, but instead greedily choose actions based on affordances computed from the current state s_t. Our training optimization parameters for these baselines are kept the same as that of VPG.

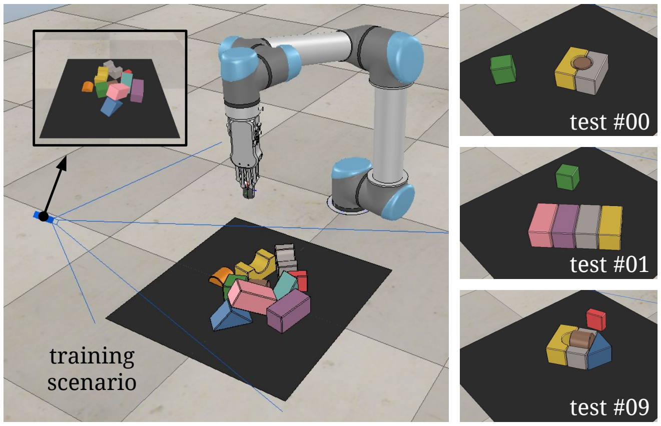

Fig. 3. **Simulation environment.** Policies are trained in scenarios with random arrangements of 10 objects (left), then evaluated in scenarios with varying degrees of clutter (10 objects, 30 objects, or challenging object arrangements). In the most challenging scenarios, adversarial clutter was manually engineered to reflect challenging real-world picking scenarios (*e.g.* tightly packed boxes, as shown on the right).

B. Evaluation Metrics

We test the methods by executing a series of tests in which the system must pick and remove objects from a table with novel arrangements of objects (as described in Sec. IV-F).

For each test, we execute n runs ($n \in \sim 10, 30$) and then evaluate performance with 3 metrics: 1) the average % completion rate over the n test runs, which measures the ability of the policy to finish the task by picking up all objects without failing consecutively for more than 10 attempts, 2) the average % grasp success rate per completion, and 3) the % action efficiency (defined as $\frac{\text{\# objects in test}}{\text{\# actions before completion}}$), which describes how succinctly the policy is capable of finishing the task. Note that grasp success rate is equivalent to action efficiency for grasping-only policies. For all of these metrics, higher is better.

We run experiments on both simulated and real-world platforms. While our main objective is to demonstrate effective VPG policies on a real robot, we also run experiments in simulation to provide controlled environments for fair evaluation between methods and for ablation studies. In experiments on both platforms, we run tests with objects placed in both random and challenging arrangements.

C. Simulation Experiments

Our simulation setup uses a UR5 robot arm with an RG2 gripper in V-REP [39] (illustrated in Fig. 3) with Bullet Physics 2.83 for dynamics and V-REP's internal inverse kinematics module for robot motion planning. Each test run in simulation was run $n = 30$ times. The objects used in these simulations include 9 different 3D toy blocks, the shapes and colors of which are randomly chosen during experiments. Most dynamics parameters are kept default except friction coefficients, which have been modified to achieve synthetic object interaction behaviors as similar as possible to that of the real-world. We did not perform any tuning of random seeds for the simulated physics in our experiments. We also simulate a statically mounted perspective 3D camera in the

TABLE I
SIMULATION RESULTS ON RANDOM ARRANGEMENTS (MEAN %)

Method	Completion	Grasp Success	Action Efficiency
Grasping-only [8]	90.9	55.8	55.8
P+G Reactive	54.5	59.4	47.7
VPG	**100.0**	**67.7**	**60.9**

environment, from which perception data is captured. RGB-D images of resolution 640 × 480 are rendered with OpenGL from the camera, without any noise models for depth or color.

Comparisons to Baselines. Our first experiment compares VPG to the two baseline methods in a simulation where 30 objects are randomly dropped onto a table. This scenario is similar to the training scenario, except it has 30 objects rather than 10, thus testing the generalization of policies to more cluttered scenarios. Results are shown in Table I. We see that VPG outperforms both baseline methods across all metrics. It is interesting to note that P+G reactive performs poorly in terms of completion rates and action efficiency. This likely due to its tendency (in the face of clutter) to continually push objects around until they are forced out of the workspace as grasping affordances remain low.

Challenging Arrangements. We also compare VPG in simulation to the baseline methods on 11 challenging test cases with adversarial clutter. Each test case consists of a configuration of 3 - 6 objects placed in the workspace in front of the robot, 3 configurations of which are shown in Fig. 3. These configurations are manually engineered to reflect challenging picking scenarios, and remain exclusive from the training procedure (described in Sec. IV-E). Across many of these test cases, objects are laid closely side by side, in positions and orientations that even an optimal grasping policy would have trouble successfully picking up any of the objects without de-cluttering first. As a sanity check, a single isolated object is additionally placed in the workspace separate from the configuration. This is just to ensure that all policies have been sufficiently trained prior to the benchmark (*i.e.* a policy is not ready if fails to grasp the isolated object).

Results are shown in Table II. From the completion results, we observe that the addition of pushing enlarges the set of the scenarios for which successful grasping can be performed. Across the collection of test cases, the grasping-only policy frequently struggles to complete the picking task (with a 0% completion rate for 5 out of the 11 test cases). We observe this be particularly true in scenarios where large cuboids are laid closely side-by-side (Fig. 3). Even when the policy does successfully complete the task, the average grasp success rates remain relatively low at 50-60%.

TABLE II
SIMULATION RESULTS ON CHALLENGING ARRANGEMENTS (MEAN %)

Method	Completion	Grasp Success	Action Efficiency
Grasping-only [8]	40.6	51.7	51.7
P+G Reactive	48.2	59.0	46.4
VPG	**82.7**	**77.2**	**60.1**

Upon the addition of pushing as an additional action primitive in the P+G reactive policy, we immediately see an increase in picking completion rates and there are no longer cases in which the policy completely fails with a 0% completion rate. While the P+G reactive policy achieves higher completion and grasp success rates than grasping-only, the average action efficiency is lower. This suggests that the policy executes a large number of pushes, many of which are not succinct and may not actually help grasping. This is expected, since P+G reactive uses binary supervision from change detection for pushing – pushing motions are not directly supervised by how well they help grasping.

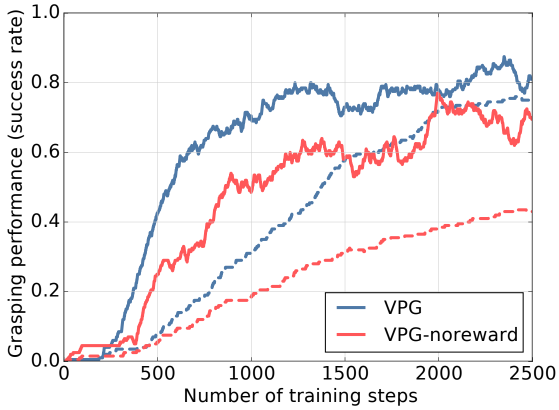

Fig. 4. Comparing performance of VPG policies trained with and without rewards for pushing. Solid lines indicate % grasp success rates (primary metric of performance) and dotted lines indicate % push-then-grasp success rates (secondary metric to measure quality of pushes) over training steps.

By enabling joint planning of pushing and grasping with VPG, we observe substantially higher completion and grasp success rates (with a 100% completion rate for 5 of the 11 test cases). The higher action efficiency also indicates that the pushes are now more succinct in how they help grasping.

No Pushing Rewards? We next investigate whether our method can learn synergistic pushing and grasping actions even without any intrinsic rewards for pushing ($R_{\mathrm{p}}(s_t, s_{t+1}) = 0$). We call this variant of our algorithm "VPG-noreward". In this more difficult setting, the pushing policy learns to effect change only through the reward provided by future grasps.

For this study, we run tests in simulation with 10 randomly placed objects. We report results with plots of grasping performance versus training steps. Grasping performance is measured by the % grasp success rate over the last $j = 200$ grasp attempts, indicated by solid lines in Fig. 4. We also report the % push-then-grasp success rates (*i.e.* pushes followed immediately by a grasp – considered successful if the grasp was successful), indicated by dotted lines. Since there is no defacto way to measure the quality of the pushing motions for how well they benefit a model-free grasping policy, this secondary metric serves as a good approximation. The numbers reported at earlier training steps (*i.e.* iteration $i < j$) in Fig. 4 are weighted by $\frac{i}{j}$. Each training step consists of capturing data, computing a forward pass, executing an action, backpropagating, and running a single iteration of experience replay (with another forward pass and backprop-

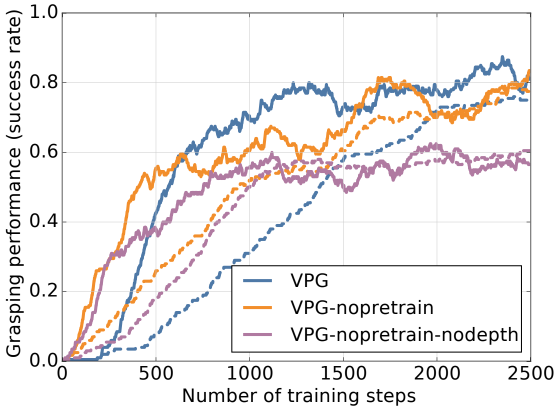

Fig. 5. Comparing performance of VPG policies initialized without weights pre-trained on ImageNet and without the depth channels of the RGB-D heightmap (*i.e.* no height-from-bottom, only color information). Solid lines indicate % grasp success rates (primary metric of performance) and dotted lines indicate % push-then-grasp success rates (secondary metric to measure quality of pushes) over training steps.

agation on a sample from the replay buffer).

From these results, we see that VPG-noreward is capable of learning effective pushing and grasping policies – achieving grasping success rates at 70-80%. We also see that it learns a pushing policy that increasingly helps grasping (note the positive slope of the dotted red line, which suggests pushes are helping future grasps more and more as the system trains). This rate of improvement is not as good as VPG, but the final performance is only slightly lower.

No ImageNet Pre-training? We trained a version of VPG (“VPG-nopretrain") without ImageNet pre-training of FCN weights (*i.e.* with only random initialization) and report its performance versus training steps in Fig. 5. Interestingly, the results suggest that ImageNet pre-training is not a major contributor to the sample efficiency of VPG nor to the final performance of the model in simulation. This could be due to the fact that the statistics of pixel patterns found in ImageNet images are different compared to that of re-projected heightmap images. The slight delay before the upward slope of the training curve could also be an artifact due to the FCNs spending early training steps to escape the ImageNet local optimum.

No Height-from-bottom Information? We trained another version of VPG (“VPG-nopretrain-nodepth") without ImageNet pre-training and without the depth channels of the RGB-D heightmap images (*i.e.* no height-from-bottom, only color information) and report its performance in Fig. 5. This modification meant that each FCN ϕ_p and ϕ_g no longer has a second DenseNet tower to compute features from the channel-wise cloned depth channels (DDD) of the heightmaps. The results show that sample complexity remains similar, but the average final grasping performance is lower by about 15%. This suggests that geometric cues from the depth (height-from-bottom) channels are important for achieving reasonable grasping performance with VPG.

Shortsighted Policies? We also investigate the importance of long-term lookahead. Our Q-learning formulation in theory enables our policies to plan long-term strategies (*e.g.* chaining multiple pushes to enable grasping, grasping to enable pushing, grasping to enable other grasps, etc.). To test the value of these strategies, we trained a shortsighted version of VPG (“VPG-myopic") where the discount factor on future rewards is smaller at $\gamma = 0.2$ (trained in simulation with 10 randomly placed objects). We evaluate this policy over the 11 hard test cases in simulation and report comparisons to our method in Table III. Interestingly, we see that VPG-myopic improves its grasping performance at a faster pace early in training (presumably optimizing for short-term grasping rewards), but ultimately achieves lower average performance (*i.e.* grasp success, action efficiency) across most hard test cases. This suggests that the ability to plan long-term strategies for sequential manipulation could benefit the overall stability and efficiency of pick-and-place.

TABLE III
COMPARISON WITH MYOPIC POLICIES (MEAN %)

Method	Completion	Grasp Success	Action Efficiency
VPG-myopic	79.1	74.3	53.7
VPG	**82.7**	**77.2**	**60.1**

D. Real-World Experiments

In this section, we evaluate the best performing variant of VPG (with rewards and long-term planning) on a real robot. Our real-world setup consists of a UR5 robot arm with an RG2 gripper, overlooking a tabletop scenario. Objects vary across different experiments, including a collection of 30+ different toy blocks for training and testing, as well as a collection of other random office objects to test generalization to novel objects (see Fig. 7). For perception data, RGB-D images of resolution 640 × 480 are captured from an Intel RealSense SR300, statically mounted on a fixed tripod overlooking the tabletop setting. The camera is localized with respect to the robot base by an automatic calibration procedure, during which the camera tracks the location of a checkerboard pattern taped onto the gripper. The calibration optimizes for extrinsics as the robot moves the gripper over a grid of 3D locations (predefined with respect to robot coordinates) within the camera field of view.

Random Arrangements. We first tested VPG on the real robot in cluttered environments with 30 randomly placed objects. Fig. 6 shows its performance versus training time in comparison to the grasping-only policy (baseline method) – where curves show the % grasp success rate over the last $m = 200$ grasp attempts (solid lines) and % push-then-grasp success rates (dotted lines) for both methods.

Interestingly, the improvement of performance early in training is similar between VPG and grasping-only. This is surprising, as one would expect VPG to require more training samples (and thus more training time) to achieve comparable performance, since only one action (either a grasp or a push) can be executed per training step. This similarity in growth of performance can likely be attributed to our method optimizing the pushing policies to make grasping easier even at a very early stage of training. While the grasping-only

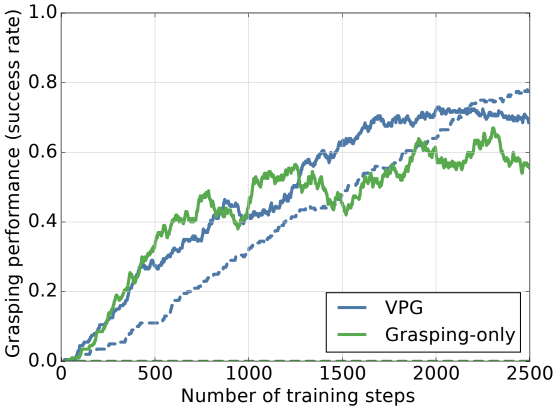

Fig. 6. Evaluating VPG in real-world tests with random 30+ object arrangements. Solid lines indicate % grasp success rates (primary metric of performance) and dotted lines indicate % push-then-grasp success rates (secondary metric to measure quality of pushes) over training steps.

policy is busy fine-tuning itself to detect harder grasps, VPG spends time learning pushes that can make grasping easier.

As expected, the grasping performance of the VPG policy surpasses that of the grasping-only policy in later training steps. Not only is the performance better, it is also less erratic. This is likely because it avoids long sequences of failed grasps, which happens occasionally for grasping-only when faced with highly cluttered configurations of objects.

This experiment also suggests that VPG is quite sample efficient – we are able to train effective pushing and grasping policies in less than 2000 transitions. At 10 seconds per action execution on a real robot, this amounts to about 5.5 hours of wall-clock training time. This is a substantial advantage over prior work on deep reinforcement learning for manipulation (*e.g*. 10 million sample transitions (10 hours of interaction time on 16 robots) for block stacking [14]).

Challenging Arrangements. We also ran experiments in the real-world comparing VPG with grasping-only on 7 challenging test cases with adversarial clutter (see examples in top row of Fig. 7). The results appear in Table IV. Note that the differences between VPG and Grasping-only are quite large in these challenging real-world cases.

TABLE IV
REAL-WORLD RESULTS ON CHALLENGING ARRANGEMENTS (MEAN %)

Method	Completion	Grasp Success	Action Efficiency
Grasping-only [8]	42.9	43.5	43.5
VPG	**71.4**	**83.3**	**69.0**

Video recordings of these experiments are provided on our project webpage [40]. They show that the VPG pushing and grasping policies perform interesting synergistic behaviors, and are more capable of efficiently completing picking tasks in cluttered scenarios in tandem than grasping-only policies.

Novel Objects. Finally, we tested our VPG models (trained on toy blocks) on a collection of real-world scenes with novel objects (examples of which are shown in the bottom row of Fig. 7). Overall, the system is capable of generalizing to sets of objects that fall within a similar shape distribution as that of the training objects, but struggles when completely new shapes or anomalies (reflective objects with no depth data) are introduced. The robot is capable of planning complex pushing motions that can de-clutter scenarios with these novel objects. We also show several video recordings of these test runs on our project webpage [40].

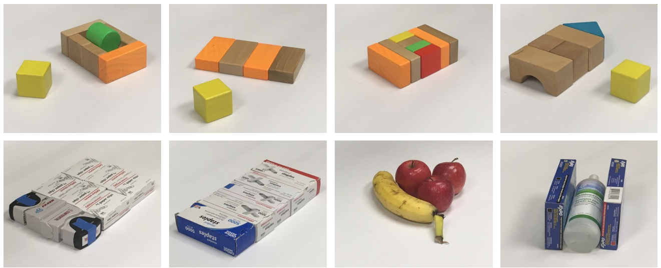

Fig. 7. **Examples of challenging arrangements** in real-world settings with toy blocks (top row) and novel objects (bottom row).

VI. Discussion and Future Work

In this work, we present a framework for learning pushing and grasping policies in a mutually supportive way. We show that the synergy between planning non-prehensile (pushing) and prehensile (grasping) actions can be learned from experience. Our method is based on a pixel-wise version of deep networks that combines deep reinforcement learning with affordance-based manipulation. Results show that our system learns to perform complex sequences of pushing and grasping on a real robot in tractable training times.

To the best of our knowledge, this work is the first to explore learning complementary pushing and grasping policies simultaneously from scratch with deep reinforcement learning. However, its limitations suggest directions for future work. First, motion primitives are defined with parameters specified on a regular grid (heightmap), which provides learning efficiency with deep networks, but limits expressiveness – it would be interesting to explore other parameterizations that allow more expressive motions (without excessively inducing sample complexity), including more dynamic pushes, parallel rather than sequential combinations of pushing and grasping, and the use of more varied contact surfaces of the robot. A second limitation is that we train our system only with blocks and test with a limited range of other shapes (fruit, bottles, etc.) – it would be interesting to train on larger varieties of shapes and further evaluate the generalization capabilities of the learned policies. Finally, we study only synergies between pushing and grasping, which are just two examples of the larger family of primitive manipulation actions, *e.g*. rolling, toppling, squeezing, levering, stacking, among others – investigating the limits of this deep reinforcement learning approach on other multi-step interactions is a significant topic for future work.

References

[1] M. T. Mason, “Mechanics and planning of manipulator pushing operations,” *IJRR*, 1986.

[2] S. Goyal, A. Ruina, and J. Papadopoulos, “Planar sliding with dry friction part 1. limit surface and moment function,” *Wear*, 1991.

[3] F. R. Hogan and A. Rodriguez, "Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics," *WAFR*, 2016.

[4] M. Bauza and A. Rodriguez, "A probabilistic data-driven model for planar pushing," in *ICRA*, 2017.

[5] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *ICRA*, 2017.

[6] C. Ignasi, D. Held, and P. Abbeel, "Policy transfer via modularity," *IROS*, 2017.

[7] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *ICRA*, 2016.

[8] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," *ICRA*, 2018.

[9] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt, "High precision grasp pose detection in dense clutter," in *IROS*, 2016.

[10] 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," *RSS*, 2017.

[11] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, 2012.

[12] A. Boularias, J. A. Bagnell, and A. Stentz, "Learning to manipulate unknown objects in clutter by reinforcement." in *AAAI*, 2015.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, 2016.

[14] I. Popov, N. Heess, T. Lillicrap, *et al.*, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv*, 2017.

[15] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *arXiv*, 2017.

[16] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *IROS*, 2016.

[17] M. Salganicoff, G. Metta, A. Oddera, and G. Sandini, *A vision-based learning method for pushing manipulation*, 1993.

[18] T. Meriçli, M. Veloso, and H. L. Akın, "Push-manipulation of complex passive mobile objects using experimentally acquired motion models," *Autonomous Robots*, 2015.

[19] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *ICRA*, 2016.

[20] D. Prattichizzo and J. C. Trinkle, "Grasping," in *Springer Handbook of Robotics*, 2008.

[21] J. Weisz and P. K. Allen, "Pose error robust grasping from contact wrench space metrics," in *ICRA*, 2012.

[22] A. Rodriguez, M. T. Mason, and S. Ferry, "From caging to grasping," *IJRR*, 2012.

[23] C. Goldfeder, M. Ciocarlie, H. Dang, and P. K. Allen, "The columbia grasp database," in *ICRA*, 2009.

[24] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge," in *ICRA*, 2017.

[25] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from rgb-d reconstructions," in *CVPR*, 2017.

[26] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *ICRA*, 2015.

[27] L. Pinto and A. Gupta, "Learning to push by grasping: Using multiple tasks for effective learning," in *ICRA*, 2017.

[28] D. Omrčen, C. Böge, T. Asfour, A. Ude, and R. Dillmann, "Autonomous acquisition of pushing actions to support object grasping with a humanoid robot," in *Humanoids*, 2009.

[29] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, CMU RI, 2010.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[32] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *CVPR*, 2017.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[36] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, 2016.

[37] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *ICML*, 2015.

[38] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *ICLR*, 2016.

[39] M. F. E. Rohmer, S. P. N. Singh, "V-rep: a versatile and scalable robot simulation framework," in *IROS*, 2013.

[40] Project webpage. [Online]. Available: http://vpg.cs.princeton.edu