NeRF in the Palm of Your Hand: Corrective Augmentation for Robotics via Novel-View Synthesis

Allan Zhou* 
Stanford

Moo Jin Kim* 
Stanford

Lirui Wang 
MIT CSAIL

Pete Florence 
Google

Chelsea Finn 
Stanford

Abstract

Expert demonstrations are a rich source of supervision for training visual robotic manipulation policies, but imitation learning methods often require either a large number of demonstrations or expensive online expert supervision to learn reactive closed-loop behaviors. In this work, we introduce SPARTN (Synthetic Perturbations for Augmenting Robot Trajectories via NeRF): a fully-offline data augmentation scheme for improving robot policies that use eye-in-hand cameras. Our approach leverages neural radiance fields (NeRFs) to synthetically inject corrective noise into visual demonstrations, using NeRFs to generate perturbed viewpoints while simultaneously calculating the corrective actions. This requires no additional expert supervision or environment interaction, and distills the geometric information in NeRFs into a real-time reactive RGB-only policy. In a simulated 6-DoF visual grasping benchmark, SPARTN improves success rates by 2.8× over imitation learning without the corrective augmentations and even outperforms some methods that use online supervision. It additionally closes the gap between RGB-only and RGB-D success rates, eliminating the previous need for depth sensors. In real-world 6-DoF robotic grasping experiments from limited human demonstrations, our method improves absolute success rates by 22.5% on average, including objects that are traditionally challenging for depth-based methods. See video results at https://bland.website/spartn.

1. Introduction

Object grasping is a central problem in vision-based control and is fundamental to many robotic manipulation problems. While there has been significant progress in top-down bin picking settings [21, 34], 6-DoF grasping of arbitrary objects amidst clutter remains an open problem, and is especially challenging for shiny or reflective objects that are not visible to depth cameras. For example, the task of grasping a wine glass from the stem shown in Figure 1 requires precise 6-DoF control (using full 3D translation and 3D rotation of the gripper) and closed-loop perception of a transparent object. Traditional 6-DoF grasping pipelines [8, 57] synthesize only one grasp pose and use a motion planner to generate a collision-free trajectory to reach the grasp [38,40,54,60]. However, the use of open-loop trajectory execution prevents the system from using perceptual feedback for reactive, precise grasping behavior. In this paper, we study how to learn closed-loop policies for 6-DoF object grasping from RGB images, which can be trained with imitation or reinforcement learning methods [58].

Imitation learning from expert demonstrations is a simple and promising approach to this problem, but is known to suffer from compounding errors [46]. As a result, complex vision-based tasks can require online expert supervision [19,46] or environment interaction [13,44], both of which are expensive and time-consuming to collect. On the other hand, offline “feedback augmentation” methods [14, 22] can be effective at combating compounding errors, but are severely limited in scope and thus far have not been ap-

*Equal contribution. Correspondence to ayz@cs.stanford.edu.

Figure 1. SPARTN is an offline data augmentation method for behavior cloning eye-in-hand visual policies. It simulates recovery in a demonstration by using NeRFs to render high-fidelity observations (right) from noisy states, then generates corrective action labels.
plied to visual observations. Other recent works have found that using eye-in-hand cameras mounted on a robot’s wrist can significantly improve the performance of visuomotor policies trained with imitation learning [17,20,35], but still do not address the underlying issue of compounding errors. We develop an approach that helps address compounding errors to improve vision-based policies, while building on the success of eye-in-hand cameras.

To improve imitation learning for quasi-static tasks like grasping, we propose a simple yet effective offline data augmentation technique. For an eye-in-hand camera, the images in each demonstration trajectory form a collection of views of the demonstration scene, which we use to train neural radiance fields (NeRFs) [37] of each scene. Then, we can augment the demonstration data with corrective feedback by injecting noise into the camera poses along the demonstration and using the demonstration’s NeRF to render observations from the new camera pose. Because the camera to end-effector transform is known, we can compute corrective action labels for the newly rendered observations by considering the action that would return the gripper to the expert trajectory. The augmented data can be combined with the original demonstrations to train a reactive, real-time policy. Since the NeRFs are trained on the original demonstrations, this method effectively “distills” the 3D information from each NeRF into the policy.

The main contribution of this work is a NeRF-based data augmentation technique, called SPARTN (Synthetic Perturbations for Augmenting Robot Trajectories via NeRF), that improves behavior cloning for eye-in-hand visual grasping policies. By leveraging view-synthesis methods like NeRF, SPARTN extends the idea of corrective feedback augmentation to the visual domain. The resulting approach can produce (i) reactive, (ii) real-time, and (iii) RGB-only policies for 6-DoF grasping. The data augmentation is fully offline and does not require additional effort from expert demonstrators nor online environment interactions. We evaluate SPARTN on 6-DoF robotic grasping tasks both in simulation and in the real world. On a previously-proposed simulated 6-DoF grasping benchmark [58], the augmentation from SPARTN improves grasp success rates by 2.8× compared to training without SPARTN, and even outperforms some methods that use expensive online supervision. On eight challenging real-world grasping tasks with a Franka Emika Panda robot, SPARTN improves the absolute average success rate by 22.5%.

2. Related Work

Robotic Grasping. Grasping is a long-studied topic in robotics [50]; see the multitude of survey articles for a complete review [2,4,26]. Most data-driven grasping systems focus on learning how to predict some parameterized “grasp” (whether a full 6-DoF pose, 2-DoF table-top position, etc.), and leave intermediate motion generation to be open-loop, handled either through motion planners or simple heuristics, e.g. [12,34,42,43,52,57]. Other works have trained closed-loop grasping policies [21,39,53,58], and bring all the benefits of closed-loop policies: including for example, the ability to avoid obstacles, to perform precise grasping without precise calibration, and to react to dynamic objects. Additionally, grasping policies are often designed for top-down (2- or 3-DoF) grasping [21,31,43], while 6-DoF grasping typically requires depth or 3D information [39,42,53,58]. And while depth sensors can provide richer geometric information about the scene and enable more precise grasping, it can also struggle on common objects such as transparent cups or shiny silverware [62]. Our method trains a reactive 6-DoF grasping policy with only RGB data. See Table 7 in the appendix for a summary of the assumptions of the most related grasping works.

Imitation learning and data augmentation. Behavior cloning is known to struggle with covariate shift: small errors cause imitation policies to fall slightly off of the data distribution and it is then difficult to correct the mistake back onto the data manifold. DAgger [46] and its variants [16,23,36] mitigate this issue by obtaining expert corrections throughout training. Alternatively, DART [29] injects noise during expert demonstration collection, which is especially effective with algorithmic experts but interferes with human demonstration collection. Previous works [14,22] have injected noise into the low-dimensional system state after data collection (in a fully offline manner), but the visual observations are left out, limiting the helpfulness of noise injection. Our method can be seen as a visual, fully-offline version of noise injection that does not require perturbing the expert during demonstration collection, using NeRF to synthetically render perturbed states post-hoc. Unlike standard image augmentations for policy learning [30,61], our method uses NeRF to learn a 3D model of the demonstration scene, which enables us to generate high-fidelity novel views for data augmentation. In addition, while standard image augmentation approaches do not modify the action labels, we leverage hand-eye coordination to calculate corrective actions for augmented observations.

NeRF for Robotics. A number of recent works have investigated applications of NeRF and related methods in robotics, including localization [63], navigation [1], dynamics modeling [7,9,33], reinforcement learning [10], and data generation for other learning-based methods [18,62]. NeRF-Supervision [62], for example, generates pixel-level correspondence to learn dense object descriptors, which are useful for manipulation tasks. For grasping, various methods have leveraged NeRF [3,18,24] for open-loop grasp synthesis. In contrast, our method uses NeRF offline to augment data for grasping and distills a reactive, real-time, RGB-only, closed-loop policy.
3. Methodology

We now introduce SPARTN, which augments an eye-in-hand robot demonstration dataset using NeRF. We first review preliminaries, then describe a method overview, followed by details of training NeRFs and augmenting corrective behavior.

3.1. Preliminaries

**Imitation learning.** In imitation learning, we assume access to a dataset of $N$ expert trajectories $D = \{\tau\}_{i=1}^N$, where each trajectory consists of a sequence of state-action pairs, $\tau = \{(s_k, a_k)\}_{k=0}^K$. In purely offline imitation learning, there exists no other primary data-collection assumptions other than this dataset $D$, i.e., no reward labels or online interactions. A standard method for this offline setting is behavior cloning (BC), which trains a policy $\pi_\theta$ to mimic the expert via supervised learning, by minimizing the objective $L(\theta) = E_{(s,a) \sim D}[\ell(\pi_\theta(s), a)]$, where $\ell$ is some loss function in the action space.

**NeRF.** Our method uses novel-view synthesis as a building block, for which we use Neural Radiance Fields [37]. For each scene, NeRF performs novel-view synthesis by training a scene-specific neural radiance field $F_\theta$ from a “training set” of posed images $\{(I_k, T_k)\}$, where each $I_k \in \mathbb{R}^{w \times h \times 3}$, $T_k \in SE(3)$. After $F_\theta$ is trained, through volume rendering NeRF can render new views of a scene from any requested pose, which can be summarized as $I = NeRF-Render(T; F_\theta)$—in particular, this works best “near” the training set of poses. Since we train many NeRFs (one per demonstration), we use an accelerated implementation of NeRF (Instant-NeRF [41]).

**Corrective Noise Augmentation.** A simple method which has been shown to improve the robustness of behavior-cloned policies is to perform corrective noise augmentation [14, 22]. The idea is to take a nominal trajectory $\tau = \{(s_k, a_k)\}_{k=0}^K$ and create a noise-distributed corrective trajectory $\tilde{\tau} = \{(\tilde{s}_k, \tilde{a}_k)\}_{k=0}^K$, where each sampled state $\tilde{s}_k$ is a noisy version of the measured state, i.e. $\tilde{s}_k \sim s_k + \epsilon$ and $\epsilon$ is sampled noise. To perform corrective feedback augmentation, $\tilde{a}_k$ is chosen such that it will return the state to the nominal trajectory: given the true inverse dynamics $f^{-1}$ of the environment, then $\tilde{a}_k = f^{-1}(\tilde{s}_k, s_{k+1})$, or compute the commanded control that can be tracked with a stabilizing controller to $s_{k+1}$. An easy way to parameterize this is by choosing the action space of the learned policy to be the input to a stabilizing controller, as in [14, 22]. Note that it is also common in the imitation and reinforcement learning literature to apply noise to inputs, which is typically interpreted as a way to regularize the policy [27, 30]. Meanwhile, in addition to potential regularization effects, the corrective noise augmentation has been interpreted to specifically reduce compounding errors [14, 22], but of course has limits if the scale of perturbations is too large or in highly non-smooth dynamics regimes. A critical limitation of prior works using corrective noise augmentation is that they have not been applied to visual observations.

3.2. Overview: Visual Corrective Augmentation

Consider the case where an agent receives partial visual observations $I$ instead of the full state of the world, and also receives direct proprioceptive state measurements, $s_{\text{robot}}$, as is commonly the case in robotics. In general, the corrective augmentation of Sec. 3.1 requires obtaining the visual observation $\hat{I}_k$ for each noisy state $\tilde{s}_k$, which could be expensive or impossible in the actual environment.

An insight of this work is that for eye-in-hand robot policies (where the visual observation comes from a camera mounted on the wrist) in static scenes, we can readily generate novel visual observations using novel-view synthesis (i.e., NeRF) without further interactions in the environment. In this setting, a primary subset of the observations are posed images $(I, T)$, together with some actions $a$.

The key intuition of our method can be grasped by considering how to perform visually corrective augmentation via NeRF, as illustrated in Figure 2. Noisy states and corrective actions $(\tilde{T}_k, \tilde{a}_k)$ can be generated for a trajectory of posed observations and actions $\tau = \{(I_k, T_k, a_k)\}_{k=0}^K$ (Sec. 3.1). The key pre-processing step is to train a trajectory-specific NeRF $F_{\theta}^\tau$ for each demonstration $\tau$ (Sec. 3.3). These trajectory-specific NeRFs enable us to render observations $\hat{I}_k$ for noisy states $\tilde{T}_k$, completing the augmentation process and resulting in visually corrective transitions $(\hat{I}_k, \hat{T}_k, \hat{a}_k)$ (Sec. 3.4). Algorithm 1 and Figure 3 overview this process.

3.3. Training NeRFs from Robot Demonstrations

SPARTN uses novel-view synthesis, in particular NeRF, to generate observations $\hat{I}_k$ for noisy states without environment interaction. We train a NeRF $F_{\theta}^\tau$ for each demonstration trajectory using the image observations $(I_1, \cdots, I_K) \in \tau$ as the training set of views. After training $F_{\theta}^\tau$, we create

```
Algorithm 1 SPARTN (Simplified overview)

Input: $D = \{\tau\}_{i=1}^N$ - expert demonstrations
Output: Augmented transition dataset $\tilde{D}$

1: $\tilde{D} \leftarrow \{}$
2: for $\tau = \{(I_k, T_k, a_k)\}_{k=1}^K \in D$ do
3: $\tilde{F}_{\theta}^\tau \leftarrow$ NeRF($\{(I_k, T_k)\}_{k=1}^K$) # train NeRF
4: for $k \in \{0, 1, \ldots, K\}$ do
5: for $i \in \{0 : N_{\text{aug}}\}$ do
6: $\epsilon \leftarrow \text{NoiseDist}(SE(3))$
7: $\tilde{T}_k \leftarrow \text{Perturb}(T_k, \epsilon)$ # perturb pose
8: $\tilde{a}_k \leftarrow \text{CorrectAction}(T_k, \tilde{T}_k, a_k)$
9: $\hat{I}_k \leftarrow \text{NeRF-Render}(T_k, \tilde{T}_k, \tilde{a}_k)$
10: $\tilde{D} \leftarrow \tilde{D} \cup \{(\hat{I}_k, \tilde{T}_k, \hat{a}_k)\}$
```
Figure 2. An illustration of how SPARTN creates augmentations from an original demonstration (in reality, this process is repeated for every available demonstration). (i): The eye-in-hand demonstration contains posed images \( \{ (I_k, T_k) \}_{k=1}^K \). (ii): We train a neural radiance field (NeRF) of the demonstration scene on the posed images. (iii): We sample perturbations around each pose to simulate noise in the demonstration, and calculate the corrective action (in magenta) that would stabilize the trajectory. (iv): We use the NeRF to render observations for the perturbed poses. The end result is augmented image-action pairs for improving behavior cloning.

Figure 3. An overview of the SPARTN training process. A NeRF is trained for each of the original demonstrations in \( D \). We use these NeRFs to generate visual corrective augmentations for each demonstration and collect them in \( \tilde{D} \). The policy \( \pi_\theta \) can be trained on \( D \) and \( \tilde{D} \) using standard behavior cloning methods.

Observations \( \tilde{I}_k \) for perturbed robot states \( \tilde{T}_k \) by rendering the view from the perturbed camera pose using \( F^j_\Theta \).

An important detail is that the end-effector reference frame used for control in demonstrations may differ from the reference frame for the camera itself, but through standard eye-in-hand calibration we can transform all visual observations used for training and augmenting the NeRFs into the cameras frame. Given a transform \( ^w T^E \) from which transforms between two frames, we simply transform all NeRF poses to the world frame: \( ^w T^C_k = ^w T^E_k E T^C \), where \( W \) is the world frame, \( E \) is the end-effector frame which changes at each step \( k \), and \( C \) is the camera (NeRF) frame statically linked to the end-effector frame, and \( E T^C \) is acquired through hand-eye calibration.

**COLMAP camera poses.** Real-world calibration error means that our camera-to-world transforms \( \{ ^W T^C_k \}_{k=1}^K \) are noisy and we obtain higher-quality NeRFs by using camera poses estimated by COLMAP [48,49]. Up to noise and a scale factor \( \beta \), the only difference from the world frame camera transforms is that COLMAP uses an arbitrary reference frame \( V \neq W \). We denote the COLMAP outputs \( \{ ^V H^C_k \}_{k=1}^K \), using \( H \) instead of \( T \) because of the difference in scale. We now introduce notation to separate the rotation and translation components of a transform:

\[
^a T^b := (\text{Rot}[^a T^b], \text{Trans}[^a T^b])
\]

Since we train the NeRFs on COLMAP’s camera poses, we must convert perturbed camera poses \( ^w T^C_k \) to COLMAP’s frame in order to render the observations. In other words, we must call \( \text{NeRF-Render}(^V H^C_k, F^j_\Theta) \), where:

\[
^V H^C_k = (\text{Rot}[^V T^C_k], \beta \text{ Trans}[^V T^C_k])
\]

Both \( \beta \) and \( ^V T^W \) can be estimated from the pairs \( \{ (^W T^C_k, ^V H^C_k) \}_{k=1}^K \), as described in Appendix D.2.

**Static Scene Assumption.** An additional consideration for robotic manipulation is that the standard NeRF formulation assumes a static scene, while manipulation tasks such as grasping will usually move objects in the scene. To address this, we apply the NeRF training and augmentation process to only the subsets of each demonstration trajectory where no robot-object interaction occurs, instead of all timesteps. For grasping, a simple and effective heuristic is to only apply SPARTN to the portions of each demonstration before the expert closes the gripper.

**Masking Out the Robot Gripper.** The robot’s gripper is often within the view of an eye-in-hand camera, which breaks the static scene assumption. To address this, we
translation. The high-fidelity perturbed image \( \varepsilon \sim \text{NeRF} \) corresponding to the noise to the measured end-effector pose, noise-distributed corrective trajectory, one can first apply simply in the global frame. In order to sample from the controller \([25]\), in which case are inputs to a global-frame stabilizing Cartesian controller (1) observations as a global-frame measured SE(3) end-effector pose \( W T_k^E \), where \( W \) refers to world-frame, and \( E \) to the end-effector frame at timestep \( k \), and (2) action as a global-frame desired SE(3) end-effector pose \( W T_k^E \). To switch to a relative action space, we adjust the action to

\[
a_k = E T_k^E = (W T_k^E)^{-1} W T_k^E = E T_k W T_k^E.
\]

(4)

To additionally formulate the corrective noise augmentation in the relative frame, we consider the SE(3)-noise \( \varepsilon \) as transforming from the noisy end-effector frame to the measured end-effector frame, i.e. \( E T_k^E := \varepsilon \). This accordingly adjusts the observation as \( W T_k^E = W T_k^E E T_k^E = W T_k^E \varepsilon \) and the relative action as:

\[
\tilde{a}_k = \tilde{E} T_k^E = (W T_k^E E T_k^E)^{-1} W T_k^E = \varepsilon^{-1} a_k.
\]

(5)

Concisely, this amounts to post-pending \( \varepsilon \) to the measured pose, and pre-pending \( \varepsilon^{-1} \) to the un-noised relative action. **Non-SE(3) actions.** Thus far we have assumed that the actions \( a \in \text{SE}(3) \) only command desired pose transforms. In practice, the action space may contain additional dimensions, for example to open and close the gripper itself. The SPARTN augmentation does not concern these aspects of the action, so the corrective actions simply copy the original action for non-SE(3) dimensions.

**Summary.** Figure 2 summarizes our augmentation procedure: given a demonstration dataset \( D \), we first train all the neural radiance fields \( \{f_{\Theta r}\}_{r \in D} \) and save the weights to disk. We then augment each transition in the original dataset with \( N_{\text{aug}} \) noisy corrective transitions produced using the process we described above, and save these transitions into an augmented dataset \( \tilde{D} \). Appendix Algorithm 2 describes the precise augmentation procedure in detail.

After the augmented dataset has been created, it can simply be combined with the original dataset to augment BC training. Various sampling strategies are possible, but in our experiments we simply construct mini-batches for BC training by sampling from the original and augmented datasets with equal probability.

### 4. Simulation Experiments

We evaluate SPARTN and related approaches in the simulated 6-DoF grasping benchmark first introduced in [58], which features a simulated Franka Emika Panda arm with a parallel-jaw gripper. Objects are placed on a table and must be lifted above a height threshold in a successful grasp. Policies receive either RGB, RGBD, or point cloud observations from a wrist camera, and control the gripper by commanding relative pose changes in 6-DoF end-effector space.

#### 4.1. Data Collection and Evaluation Protocol

We follow the training and evaluation procedure from [58]. The training dataset includes 2,500 demonstrations of grasping 1,500 ShapeNet [6] objects. Demonstrations are up to 20 timesteps and are generated by trajectory...
optimization to precomputed grasps from the ACRONYM dataset [11]. Policies are evaluated on grasping held-out objects from the YCB [5] or ShapeNet (SN) [6] in a simulated 6-DoF grasping benchmark [58]. We bold the best offline RGB-only results, though we include online and non-RGB methods for comparison. Offline requires additional environment interactions unlike Online. DART\(^1\) requires a special demonstration collection setup. SPARTN outperforms other offline RGB-only methods, while RL (GA-DDPG) performs best overall while requiring millions of interactions. We calculate average success rates and standard error over 4 random seeds. *Italicized* success rates were reported in prior work\(^2\) [58].

### 4.2. Comparisons

We compare SPARTN against other approaches ranging from simple behavior cloning to online-supervised imitation and reinforcement learning:

1. **Behavior Cloning (BC):** Trains policies via supervised learning on the demonstration dataset. DART [29]: Introduces a modified demonstration setup where a continuous Gaussian noise is injected into the robot’s state during the expert demonstration, then trains policies on the modified demonstrations with BC.

2. **Homography Augmentation (HA):** A simplification of SPARTN where perturbations can be 3D rotations, but not translations, of the camera. For pure rotation, we can calculate the homography transform for rendering the rotated image without NeRF. Augmented actions are computed similarly to SPARTN.

3. **DAgger [46]:** An online-supervised method where a policy is first trained using BC on offline demos, and then the expert provides action labels for states from the policy’s rollouts throughout further policy training. GA-DDPG [58]: Jointly trains policies via BC and fine-tunes them with reinforcement learning (DDPG [51]).

Of the methods considered, DAGger and GA-DDPG require online environment interaction and supervision from an expert or reward function, respectively. The other methods, including SPARTN, train only on pre-collected demonstration datasets. Since the DART data collection setup noisily perturbs the robot as the expert demonstrates, DART can interfere with human expert demonstration, though this challenge does not arise here with our simulated expert.

**Training details.** SPARTN follows the BC (RGB) hyperparameters of [58], where policies consist of a ResNet-18 [15] image encoder followed by an MLP. Appendix C.1 describes the complete training details.

### 4.3. Results

Table 1 shows grasping success rates on held-out objects from either the YCB or ShapeNet (SN) datasets. SPARTN significantly outperforms the other two offline-supervised

<table>
<thead>
<tr>
<th>Supervision</th>
<th>Method</th>
<th>Input YCB SR(%)</th>
<th>SN SR(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline</td>
<td>BC</td>
<td>RGB 24.9 ± 2.4</td>
<td>57.4 ± 0.2</td>
</tr>
<tr>
<td></td>
<td>DART(^1)</td>
<td>RGB 51.2 ± 3.2</td>
<td>57.6 ± 2.2</td>
</tr>
<tr>
<td></td>
<td>Point cloud</td>
<td>RGB 46.3</td>
<td>45.3</td>
</tr>
<tr>
<td></td>
<td>SPARTN (^1)</td>
<td>RGB 54.6 ± 5.6</td>
<td>65.0 ± 6.5</td>
</tr>
<tr>
<td>Online(^*)</td>
<td>DAgger</td>
<td>RGB 52.3 ± 2.2</td>
<td>54.1 ± 0.2</td>
</tr>
<tr>
<td></td>
<td>Point cloud</td>
<td>RGB 77.2</td>
<td>75.6</td>
</tr>
</tbody>
</table>

Table 2. Comparing SPARTN to DART when controlling for an equal number of augmented transitions or an equal number of expert demos. This shows that SPARTN benefits from cheaply generating more transitions without having to query the expert.

<table>
<thead>
<tr>
<th>Policy Input</th>
<th>Augmentation YCB SR(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>None 28.9 ± 2.4</td>
</tr>
<tr>
<td>Image + Robot state</td>
<td>None 25.6 ± 0.5</td>
</tr>
<tr>
<td>Image + Robot state</td>
<td>SPARTN(−Image) 1.7 ± 0.3</td>
</tr>
<tr>
<td>Image + Robot state</td>
<td>SPARTN 75.6 ± 2.4</td>
</tr>
</tbody>
</table>

Table 3. If we give (robot) state as input to the policy, we can ablate SPARTN by only augmenting the state and action and not the image: SPARTN(−Image). This breaks the correct image-action correspondence and performs poorly relative to full SPARTN.

Figure 5. Example tasks in the simulated 6-DoF grasping benchmark, first introduced in [58]. There are \(\sim 1,500\) ShapeNet objects in training demonstrations, and held out YCB objects for evaluation. A camera mounted to the Franka Panda robot’s arm provides observations, and the policy controls the 6-DoF end-effector pose.
Figure 6. Real-world grasping environments. In each environment, the task is to grasp the labeled target object in a particular way. The target objects have various geometric shapes and exhibit a diverse range of characteristics, including reflectiveness, transparency, and radial symmetry.

RGB approaches. Since DART injects noise during expert collection, the amount of augmentation is limited by the number of demonstrations the expert can collect. Meanwhile, SPARTN can cheaply generate an arbitrary number of augmented examples on top of the existing demonstration dataset, leading to more data diversity without any additional effort from the expert.

SPARTN is especially beneficial on YCB, which contains the more realistic objects. It outperforms DAgger (RGB) on YCB, perhaps because DAgger’s nonstationary data distribution presents a more challenging learning problem. SPARTN is also comparable to DAgger (Point cloud) here, which significantly closes the gap between RGB-only and the best depth-based methods.

Controlling transition vs demo quantity. For the same number of expert demonstrations, SPARTN generates many more augmented transitions than DART since it does not need to query the expert. Table 2 shows that if we limit SPARTN to providing an equal number of augmented transitions as DART, the relative improvement vanishes. As expected, the advantage of SPARTN is in cheaply generating many augmentations, but does not provide more benefit than DART per-transition.

Augmenting state without image. SPARTN generates augmentations by (1) perturbing the robot state, (2) using NeRF to render the image observation from the new state, and (3) computing the corrective action. To test the importance of the NeRF component, we ran a version of our method called SPARTN(−Image) where we ablate only (2). Table 3 shows that this seriously degrades performance, since it breaks the proper image-action correspondence, while robot state alone is insufficient to solve the task.

5. Real-World Experiments

The simulation benchmark results show that SPARTN can improve grasping generalization in imitation learning without online supervision. Here, we verify that SPARTN can enable real-world robotic grasping of challenging objects from limited human demonstration. See the website for video results.

5.1. Experimental details

Robot setup. The robotic manipulator is a Franka Emika Panda robot arm with a wrist-mounted consumer grade webcam. Policies take images of size $256 \times 256$ as input and output a 6-DoF action representing the desired change in end-effector position and orientation, as well as a binary open/close gripper action. We use a Cartesian impedance controller to command the pose changes at a frequency of 4 Hz. We task the robot to grasp a target object in eight different environments depicted in Figure 6. The target objects include natural shapes that are common in the real world and exhibit a diverse range of attributes, such as reflectiveness, transparency, and radial symmetry.

Comparisons and Evaluation. In each environment, we collect a small number of expert grasping demonstrations with a virtual reality controller. Because DART is difficult to use with human demonstration, we compare SPARTN to a vanilla BC policy on the same set of demonstrations. Policies are evaluated with the same objects seen in the demonstrations. Initial object and robot configurations are randomized during both data collection and evaluation. Appendix D describes training details for each method.

5.2. Results

Table 4 shows grasping success rates in the eight real-world environments. Quantitatively, SPARTN policies outperform the baseline BC policies across the board, on average achieving an absolute 22.5% increase in success rate. Figure 7 shows qualitative differences in performance between the BC and SPARTN policies. SPARTN generally exhibits more reactive behaviors than the baseline policy: it navigates towards the target object better while occasionally avoiding obstacles, executes actions with greater precision,
Figure 7. Sample real-world policy rollouts illustrating how SPARTN succeeds (green) in cases where BC fails (red). Top left: While BC fails to reach the steel bottle, SPARTN successfully reaches and grasps it. Top right: While BC collides into the orange bottle (a distractor object), SPARTN takes a more rounded path to avoid it before grasping the white Lubriderm bottle. Bottom left: BC fails to recover after a missed grasp, while SPARTN successfully reattempts the grasp after failing the first time. Bottom right: SPARTN operates with higher precision than BC and successfully completes a difficult fork grasping task.

and even reattempts the grasp more successfully after an initial miss. In some cases, the differences are stark: for instance, SPARTN may successfully move toward and grasp the target object while the baseline fails to even reach it. We present further analysis of policy rollouts in Appendix D.4, showing how SPARTN qualitatively performs better than BC even in cases where both methods fail to grasp the target object. The website’s videos of real-world policy rollouts illustrate all of these differences, revealing that SPARTN induces important reactive closed-loop behaviors that enable the manipulator to successfully execute grasps in the real world.

Object Generalization. We additionally evaluate whether SPARTN enables greater generalization to held-out target objects outside the distribution of objects seen at training time. We train a SPARTN policy and BC policy on a training set consisting of 5 grasping demonstrations for each of 20 target objects (Figure 8, left), and then evaluate each policy against 5 held-out test objects (Figure 8, middle). The results, shown in Table 5, indicate that SPARTN improves object generalization in most cases, as the corrective augmentations enhance the policy’s robustness.

6. Conclusion

We introduce SPARTN, which augments eye-in-hand demonstrations with perturbed visual observations and corrective actions. Our augmentation leverages novel-view synthesis, in particular NeRF, to produce these augmentations during training-time. SPARTN can improve behavior cloning training of robust, real-time, and closed-loop 6-DoF visual control policies. We show that SPARTN-trained policies outperform other offline-supervised methods in a simulated 6-DoF grasping generalization benchmark. Our policies can also perform on par with imitation methods that require depth information and online supervision. We verify that SPARTN can train policies to grasp a variety of objects in the real world from limited human demonstrations.

Despite its strong performance, SPARTN also has some limitations. First, SPARTN is limited to tasks with static scenes like grasping: extending to a broader set of manipulation tasks would require effective view synthesis for dynamic scenes. Second, training a neural radiance field for every demonstration before training is computationally expensive, a limitation that may be mitigated through amortized NeRF models [55, 59, 64]. Finally, the noise distribution must be tuned for a given platform, e.g. it is tuned separately for the simulated and real experiments. An interesting direction for future work is to use policy actions to generate the noise, which would result in a fully offline variant of DAgger [46] that uses NeRF as a simulator.

7. Acknowledgements

We thank Jimmy Wu, Kaylee Burns, Bohan Wu, and Suraj Nair for technical advice on various aspects of the real robot setup, and Archit Sharma for helpful conceptual discussions. This project was funded by ONR grant N00014-21-1-2685. AZ acknowledges the support of the NSF Graduate Research Fellowship. CF is a fellow of the CIFAR Learning in Machines and Brains Program.
References


[64] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 8
A. Full Algorithm

Algorithm 2: SPARTN: Corrective Augmentation via NeRF

Input: \( D = \{ \tau \}_{\tau=1}^{N} \) - expert demonstrations
Input: \( ETC \) - Transform between camera and end-effector
Input: \( M \in \{0,1\}^{w \times h} \) - mask for robot gripper
Output: Augmented transition dataset \( \hat{D} \)

1: \( \hat{D} \leftarrow \{ \}
2: \textbf{for } \tau \in \{ (I_k, W_k T_k^E), (ET_k)^K \}_{k=1}^{K} \in D \textbf{ do }
3: \quad \# EE pose \rightarrow camera pose
4: \quad W_k T_k^C \leftarrow W_k T_k E T_k^C \text{, for } k = 0, \ldots, K
5: \quad \{V_k T_k^C\}_{k=1}^{K} \leftarrow \text{COLMAP}(I_0, \ldots, I_K)
6: \quad \beta \leftarrow \text{SOLVE}(\text{Eq. 8}) \# Estimate \beta
7: \quad V_k T_k^W = V_k T_k^C \left( V_k T_k^C \right)^{-1}, \text{ for } k = 0, \ldots, K
8: \quad \# train NeRF
9: \quad F_{\theta_{k}} \leftarrow \text{NeRF}(\{ (I_k, V_k T_k^C)\}_{k=1}^{K}, M)
10: \textbf{for } k \in \{0,1, \ldots, K\} \textbf{ do }
11: \quad \textbf{for } i \in [N_{\text{aug}}]
12: \quad \varepsilon \leftarrow \text{NoiseDist}(SE(3))
13: \quad W_k T_k^E \leftarrow W_k T_k^E \varepsilon \# perturbed end-effector pose
14: \quad \hat{E}_{k} T_{k}^{E} \leftarrow \varepsilon^{-1} E_{k} T_{k}^{E} \# corrective relative action
15: \quad W_k T_k^C \leftarrow W_k T_k^E E_{k} T_{k}^{E} \# perturbed camera pose
16: \quad V_k T_k^C \leftarrow V_k T_k^W W_k T_k^C
17: \quad \hat{I}_k \leftarrow \text{NeRF-Render}(V_k T_k^C, F_{\theta_{k}})
18: \quad \# splice gripper into rendered image
19: \quad \hat{I}_k \leftarrow \gamma M \odot \hat{I}_k + M \odot I_k
20: \quad \hat{D} \leftarrow \hat{D} \cup \{ (\hat{I}_k, W_k T_k^E), (ET_k) \}

Table 6. Ablating the effect of standard image augmentation on each method for RGB policies. Average success rates and standard errors are calculated over four seeds.

<table>
<thead>
<tr>
<th>Method</th>
<th>Image aug.</th>
<th>YCB SR(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>Without</td>
<td>25.3 ± 1.6</td>
</tr>
<tr>
<td></td>
<td>With</td>
<td>28.9 ± 2.4</td>
</tr>
<tr>
<td>DART</td>
<td>Without</td>
<td>51.2 ± 3.2</td>
</tr>
<tr>
<td></td>
<td>With</td>
<td>48.6 ± 2.8</td>
</tr>
<tr>
<td>HA</td>
<td>Without</td>
<td>23.9 ± 3.9</td>
</tr>
<tr>
<td></td>
<td>With</td>
<td>29.7 ± 2.4</td>
</tr>
<tr>
<td>SPARTN</td>
<td>Without</td>
<td>68.3 ± 5.6</td>
</tr>
<tr>
<td></td>
<td>With</td>
<td>74.7 ± 2.4</td>
</tr>
</tbody>
</table>

For behavior cloning, we use the same 3D point-matching loss as [58]. We optimize the objective using the Adam optimizer with batch size 100 for 100,000 steps. We train only the MLP (and not the pre-trained ResNet) for the first 20,000 steps, since “freeze-then-train” fine-tuning techniques have been shown to be more robust [28]. After 20,000 steps, the ResNet is unfrozen and the entire network is trained as usual.

We also apply standard image augmentations (random crop and color jitter) during training for all RGB-based methods. We ablate the effect of these image augmentations in Table 6. The results show that the image augmentations modestly improve the performance of most methods, but do not change the relative ordering of each method (BC vs DART vs HA vs SPARTN).

C.2. SPARTN Details

We use Instant-NGP [41] to train a NeRF for each of the 2,500 demonstration scenes. We train each NeRF for 3,500 steps, which takes 30 seconds on an NVIDIA GeForce RTX 2080 Ti. To reduce total training time, we train the NeRFS in parallel: with 4 GPUs, we can train all 2,500 NeRFS in ~ 7 hours. Because camera calibration is exact in simulation, we use the world-frame camera poses given by calibration instead of COLMAP to train NeRF.

We then create \( N_{\text{aug}} = 100 \) augmented transitions from each original transition and save the augmented dataset to disk before BC training. To sample the perturbations \( \varepsilon \sim \text{NoiseDist}(SE(3)) \), we parameterize the rotations in terms of Euler angles \((\phi, \theta, \varphi)\) and uniformly sample both rotation and translation parameters:

\[
(\phi, \theta, \varphi), (t_x, t_y, t_z) \sim \mathcal{U}(-\alpha, \alpha), \mathcal{U}(-\beta, \beta)
\]

\[
\varepsilon := (R(\phi, \theta, \varphi), (t_x, t_y, t_z))
\]

In simulation, we set \( \alpha = 0.2 \) radians and \( \beta = 3 \) mm. Following the DART hyperparameters in [58], we only augment timesteps \( 5 \) – \( 13 \) of each demonstration. Appendix B contains samples of SPARTN’s perturbed observations rendered via NeRF.

B. Sample Perturbed Observations

As described in the main text, image observations \( \hat{I} \) for perturbed camera poses are rendered via novel-view synthesis techniques, in particular NeRF. The section “Example NeRF renders from augmented poses” in the website shows videos of the rendering output for both simulation and real-world augmentation.

C. Simulation Details

C.1. Training for Simulation Experiments

We train all of our RGB policies (SPARTN, DART, and HA) closely following the architectural and optimization hyperparameters of [58]. The image encoder is a ResNet-18 [15] pretrained on ImageNet [47], followed by a 3-layer MLP with 512 hidden units each and ReLU activations. The MLP outputs the predicted action as a 6-D vector of predicted translations and Euler angles. The final layer has a tanh activation to scale the output between \([-1, 1]\), then each dimension is scaled to match the environment’s action bounds.
D. Real-World Details

D.1. Policy Training

For both BC and SPARTN, the image encoder for each policy is a four-layer convolutional network followed by two feedforward layers, and uses batch normalization and ReLU activation functions. The image embedding is fed into a three-layer ReLU MLP policy head, which outputs the predicted action. We train the policy from scratch with mean squared error, using Adam with learning rate $5 \times 10^{-4}$ and a batch size of 64. No image augmentations like random crop are applied.

D.2. COLMAP and NeRF Training

In the real world, we train our NeRF using camera poses $\{V^C_h\}_{k=1}^K$ estimated by COLMAP [48, 49]. From the camera calibration, we also have pose transforms $\{W^C_h\}_{k=1}^K$ which are too noisy to train a good NeRF from, but will allow us to convert between world and COLMAP frames. Conversion is necessary because we want to render from perturbed camera poses $W^C_k$, but we must call NeRF-Render using the transform $V^C_h$ (COLMAP’s reference frame) instead. From Eq. 2, we can do this conversion if we have the transform $V^W$ and the scale factor $\beta$ which distinguishes $T$ and $H$.

We estimate both quantities using the $(W^C_k, V^C_h)$ pairs in each demonstration. Recalling that $a T b = (b T a)^{-1}$, we have for any $j \neq k$:

$$\beta \text{ Trans } [C T^W_k C T^C] = \text{ Trans } [C H^V_j C H^C_k]$$

This leads to an overconstrained linear system in one unknown $\beta$, which can be solved with linear regression. After solving for $\beta$, we can rescale from $H \rightarrow T$:

$$V^C_k := (\text{Rot } [C H^V_k], (1/\beta) \text{ Trans } [C H^V_k])$$

Now solving for $V^T$ is simple:

$$V^T_k = V^C_k (W^C_k)^{-1}$$

In theory, $V^T_k$ is constant for all $k$, but in practice there is noise from both COLMAP and the real-world pose estimates, so we simply compute $V^T_k$ separately for each $k$.

D.3. SPARTN Details

To improve NeRF quality for SPARTN, we program the robot to collect a few images of the scene from a fixed set of poses before the start of each demonstration. This automatic process is only used to improve COLMAP’s pose estimation and the subsequent NeRF training. For SPARTN, we generate $N_{aug} = 50$ augmented transitions from each original transition in the demonstrations. We sample perturbations $\epsilon$ according to Eq. 6 with $\alpha = 0.05$ radians and mean squared error, using Adam with learning rate $5 \times 10^{-4}$ and a batch size of 64. No image augmentations like random crop are applied.

D.4. Additional Qualitative Analysis

Figure 9 shows sample evaluation trials where both SPARTN and BC policies fail, given the same initial object and end-effector configurations. Even in these failure cases, SPARTN qualitatively performs better than BC, nearly grasping the object in some cases, and dropping the object soon after grasping in other cases. See the videos on the website for additional qualitative results.

D.5. Real-World Environments

The real-world environments shown in Figure 6 each include a single target object to grasp among other distractor objects. The initial positions of all objects and the robotic end-effector are randomized while collecting expert demonstrations and during test time. The initial orientations of the objects are also randomized such that all policies must perform end-effector orientation control to complete the task. To make a fair comparison, we evaluate different policies given the same set of initial configurations (as shown in Figure 7, Figure 9, and the videos on the website); we do this by resetting the robot and environment to the same conditions and alternating the rollouts of different policies before

Figure 9. Sample real-world policy rollouts illustrating cases where both BC and SPARTN policies fail. Top left: BC fails to reach the thin box, while SPARTN successfully reaches it and nearly completes the grasp. Top right: While BC completely misses the white tape roll, SPARTN makes contact but fails to grasp it. Bottom left: BC fails to reach the steel water bottle, while SPARTN contacts it but fails to grasp it. Bottom right: BC reaches the Lubriderm bottle over but knocks it over before grasping it, while SPARTN nearly completes the grasp as the bottle barely slips away from the robot’s fingers.

$\beta = 0.4$ mm. The website shows sample SPARTN observations rendered by NeRF. Aside from using the augmented dataset, SPARTN policies are trained using the same BC architecture and training hyperparameters described in Appendix D.1.
Table 7. Comparison of our approach with related grasping work. SPARTN is the only approach to learn closed-loop 6-DoF grasping policies from only RGB inputs.

<table>
<thead>
<tr>
<th>Method</th>
<th>No Depth</th>
<th>Full 6-DoF</th>
<th>Closed-Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>[34, 45]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[53]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[18, 24, 40, 54]</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[21, 32]</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>[43]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[39]</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>[56, 58]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPARTN (ours)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

moving on to the next test configuration. Rollouts are considered successful if the robot grasps the target object and lifts it steadily into the air. We describe the details of each environment below:

- **Banana**: The environment includes an artificial banana resting on a white ceramic plate, a red cup, an artificial avocado, an artificial tomato, and a plastic fork and knife resting on a blue cloth.

- **Thin box**: The environment includes a thin brown box resting on a thicker white toy box.

- **Steel water bottle**: The environment includes a steel water bottle, a red water bottle, and an aluminum can. The steel water bottle is highly reflective.

- **Wine glass**: The environment includes a wine glass, red water bottle, and blue water bottle all placed upside-down and leaning against a brown box. The wine glass is transparent.

- **Lubriderm bottle**: The environment includes a white Lubriderm bottle, an orange Neutrogena bottle, and a small white container of moisturizer.

- **White tape roll**: The environment includes a roll of white tape resting on a roll of blue tape, as well as a pair of black scissors. The rolls of tape are radially symmetric.

- **Tupperware**: The environment includes a plastic tupperware container with a red lid, as well as three artificial food items: a donut, a cookie, and a slice of pizza.

- **Fork**: The environment includes a fork resting on top of a plastic tupperware container. The fork’s location and orientation relative to the surface of the tupperware container is fixed, but the position of the entire assembly is randomized.