2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

# Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World

Josh Tobin[1], Rachel Fong[2], Alex Ray[2], Jonas Schneider[2], Wojciech Zaremba[2], Pieter Abbeel[3]

***Abstract*— Bridging the 'reality gap' that separates simulated robotics from experiments on hardware could accelerate robotic research through improved data availability. This paper explores *domain randomization*, a simple technique for training models on simulated images that transfer to real images by randomizing rendering in the simulator. With enough variability in the simulator, the real world may appear to the model as just another variation. We focus on the task of object localization, which is a stepping stone to general robotic manipulation skills. We find that it is possible to train a real-world object detector that is accurate to $1.5\,\mathrm{cm}$ and robust to distractors and partial occlusions using only data from a simulator with non-realistic random textures. To demonstrate the capabilities of our detectors, we show they can be used to perform grasping in a cluttered environment. To our knowledge, this is the first successful transfer of a deep neural network trained *only* on simulated RGB images (without pre-training on real images) to the real world for the purpose of robotic control.**

## I. INTRODUCTION

Robotic learning in a physics simulator could accelerate the impact of machine learning on robotics by allowing faster, more scalable, and lower-cost data collection than is possible with physical robots. Learning in simulation is especially promising for building on recent results using deep reinforcement learning to achieve human-level performance on tasks like Atari [29] and robotic control [26], [41]. Deep reinforcement learning employs random exploration, which can be dangerous on physical hardware. It often requires hundreds of thousands or millions of samples [29], which could take thousands of hours to collect, making it impractical for many applications. Ideally, we could learn policies that encode complex behaviors entirely in simulation and successfully run those policies on physical robots with minimal additional training.

Unfortunately, discrepancies between physics simulators and the real world make transferring behaviors from simulation challenging. *System identification*, the process of tuning the parameters of the simulation to match the behavior of the physical system, is time-consuming and error-prone. Even with strong system identification, the real world has *unmodeled physical effects* like nonrigidity, gear backlash, wear-and-tear, and fluid dynamics that are not captured by current physics simulators (though learning techniques may help bridge this gap [33]). Furthermore, *low-fidelity simulated sensors* like image renderers are often unable to reproduce the richness and noise produced by their real-world counterparts. These differences, known collectively as the *reality gap*, form the barrier to using simulated data on real robots.

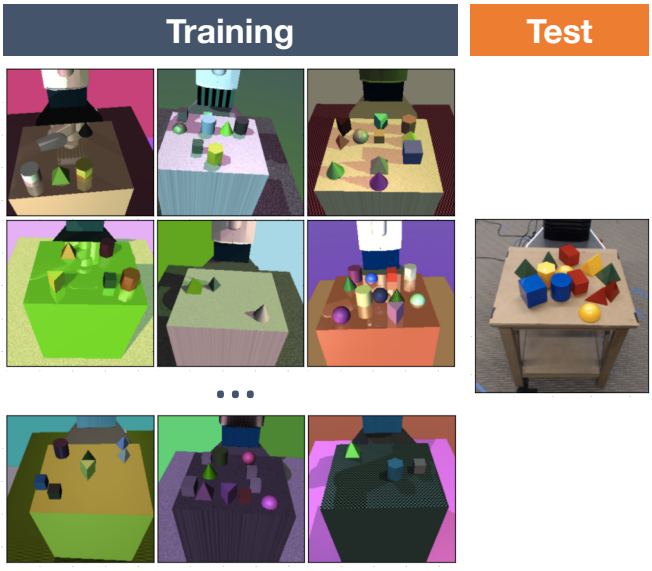


Fig. 1. Illustration of our approach. An object detector is trained on hundreds of thousands of low-fidelity rendered images with random camera positions, lighting conditions, object positions, and non-realistic textures. At test time, the same detector is used in the real world with no additional training.

This paper explores *domain randomization*, a simple but promising method for addressing the reality gap. Instead of training a model on a single simulated environment, we randomize the simulator to expose the model to a wide range of environments at training time. The purpose of this work is to test the following hypothesis: if the variability in simulation is significant enough, models trained in simulation will generalize to the real world with no additional training.

Though in principle domain randomization could be applied to any component of the reality gap, we focus on the challenge of transferring from low-fidelity simulated camera images. Robotic control from camera pixels is attractive due to the low cost of cameras and the rich data they provide, but challenging because it involves processing high-dimensional input data. Recent work has shown that supervised learning with deep neural networks is a powerful tool for learning generalizable representations from high-dimensional inputs [24], but deep learning relies on a large amount of labeled data. Labeled data is difficult to obtain in the real world for precise robotic manipulation behaviors, but it is easy to generate in a physics simulator.

We focus on the task of training a neural network to

[1]OpenAI and UC Berkeley EECS, josh@openai.com
[2]OpenAI, {rfong, aray, jonas, woj}@openai.com
[3]OpenAI, UC Berkeley EECS & ICSI, pieter@openai.com

detect the location of an object. Object localization from pixels is a well-studied problem in robotics, and state-of-the-art methods employ complex, hand-engineered image processing pipelines (e.g., [7], [6], [47]). This work is a first step toward the goal of using deep learning to improve the accuracy of object detection pipelines. Moreover, we see sim-to-real transfer for object localization as a stepping stone to transferring general-purpose manipulation behaviors.

We find that for a range of geometric objects, we are able to train a detector that is accurate to around $1.5\,\mathrm{cm}$ in the real world using only simulated data rendered with simple, algorithmically generated textures. Although previous work demonstrated the ability to perform robotic control using a neural network pretrained on ImageNet and fine-tuned on randomized rendered pixels [40], this paper provides the first demonstration that domain randomization can be useful for robotic tasks requiring precision. We also provide an ablation study of the impact of different choices of randomization and training method on the success of transfer. We find that with a sufficient number of textures, pre-training the object detector using real images is unnecessary. To our knowledge, this is the first successful transfer of a deep neural network trained *only* on simulated RGB images to the real world for the purpose of robotic control.

## II. RELATED WORK

### A. Object detection and pose estimation for robotics

Object detection and pose estimation for robotics is a well-studied problem in the literature (see, e.g., [5], [6], [7], [12], [47], [53]). Recent approaches typically involve offline construction or learning of a 3D model of objects in the scene (e.g., a full 3D mesh model [47] or a 3D metric feature representation [6]). At test time, features from the test data (e.g., Scale-Invariant Feature Transform [SIFT] features [14] or color co-occurrence histograms [12]) are matched with the 3D models (or features from the 3D models). For example, a black-box nonlinear optimization algorithm can be used to minimize the re-projection error of the SIFT points from the object model and the 2D points in the test image [5]. Most successful approaches rely on using multiple camera frames [7] or depth information [47]. There has also been some success with only monocular camera images [5]. Neural network-based perception has also been explored [25], but collecting a large enough training set can be challenging.

Compared to our method, traditional approaches require less extensive training and take advantage of richer sensory data, allowing them to detect the full 3D pose of objects (position and orientation) without any assumptions about the location or size of the surface on which the objects are placed. However, our approach avoids the challenging problem of 3D reconstruction, and employs a simple, easy to implement deep learning-based pipeline that may scale better to more challenging problems.

### B. Domain adaptation

The computer vision community has devoted significant study to the problem of adapting vision-based models trained in a source domain to a previously unseen target domain (see, e.g., [11], [16], [17], [23]). Approaches include re-training the model in the target domain (e.g., [54]), adapting the weights of the model based on the statistics of the source and target domains (e.g., [27]), learning invariant features between domains (e.g., [50]), and learning a mapping from the target domain to the source domain (e.g., [46]). Researchers in the reinforcement learning community have also studied the problem of domain adaptation by learning invariant feature representations [15], adapting pretrained networks [38], and other methods. See [15] for a more complete treatment of domain adaptation in the reinforcement learning literature.

In this paper we study the possibility of transfer from simulation to the real world *without* performing domain adaptation.

### C. Bridging the reality gap

Previous work on leveraging simulated data for physical robotic experiments explored several strategies for bridging the reality gap.

One approach is to make the simulator closely match the physical reality by performing system identification and using high-quality rendering. Though using realistic RGB rendering alone has had limited success for transferring to real robotic tasks [19], incorporating realistic simulation of depth information can allow models trained on rendered images to transfer well to the real world [35]. Combining data from high-quality simulators with other approaches like fine-tuning can also reduce the number of labeled samples required in the real world [37].

Unlike these approaches, ours allows the use of low-quality renderers optimized for speed and not carefully matched to real-world textures, lighting, and scene configurations.

Other work explores using domain adaptation to bridge the reality gap. It is often faster to fine-tune a controller learned in simulation than to learn from scratch in the real world [9], [21]. In [13], the authors use a variational autoencoder trained on simulated data to encode trajectories as a low-dimensional latent code. A policy learned on real data can overcome the reality gap by choosing latent codes via exploration that correspond to the desired physical behavior. In the evolutionary robotics literature, the work of Cully et al. [8] demonstrates that exploration in the real world can be more efficient with prior knowledge from a simulator.

Domain adaptation has also been applied to robotic vision. In [56], the authors show that modularity between the perception system and control system can aid transferability. Rusu et al. [39] find that using the progressive network architecture has better sample efficiency than fine-tuning or training in the real-world alone. In [49], the authors learn a correspondence between domains that allows the real images to be mapped into a space understood by the model. While the preceding approaches require reward functions or labeled data, Mitash and collaborators [28] pre-train an object detector using realistic rendered images to bootstrap an automated learning

learning process that does not require manually labeling data and uses only around 500 real-world samples.

A related idea, *iterative learning control*, uses real-world data to improve the dynamics model used to determine the optimal control behavior, rather than using real-world data to improve the controller directly. Iterative learning control starts with a dynamics model, applies the corresponding control behavior on the real system, and then closes the loop by using the resulting data to improve the dynamics model. Iterative learning control has been applied to a variety of robotic control problems, from model car control (e.g., [1] and [10]) to surgical robotics (e.g., [51]). Similar ideas have been explored in the evolutionary robotics literature, such as in the work of Koos et al. [22].

Domain adaptation and iterative learning control are important tools for addressing the reality gap, but in contrast to these approaches, ours requires no additional training on real-world data. Our method can also be combined easily with most domain adaptation techniques.

Several authors have previously explored the idea of using domain randomization to bridge the reality gap.

In the context of physics adaptation, Mordatch and collaborators [30] show that training a policy on an ensemble of dynamics models can make the controller robust to modeling error and improve transfer to a real robot. Similarly, in [2], the authors use a simulator with randomized friction and action delay and find behaviors transfer to the real world.

Rather than relying on controller robustness, Yu et al. [55] use a model trained on varied physics to perform system identification using online trajectory data, but their approach is not shown to succeed in the real world. Rajeswaran et al. [36] explore different training strategies for learning from an ensemble of models, including adversarial training and adapting the ensemble distribution using data from the target domain, but also do not demonstrate successful real-world transfer.

Earlier work in evolutionary robotics also uses domain randomization to encourage sim-to-real transfer. In [18], the authors suggested that transferability can be achieved by randomly varying the items in the *implementation set* – model parameters that are not essential to the controller achieving near-optimal performance. Our work can be interpreted in this framework by considering the rendering aspects of the simulator (lighting, texture, etc) as part of the implementation set.

Researchers in computer vision have used 3D models as a tool to improve performance on real images since the earliest days of the field (e.g., [32]). More recently, 3D models have been used to augment training data to aid transferring deep neural networks between datasets and prevent over-fitting on small datasets for tasks like viewpoint estimation [43] and object detection [45], [31]. Recent work has explored using only synthetic data for training 2D object detectors (i.e., predicting a bounding box for objects in the scene). In [34], the authors find that by pretraining a network on ImageNet and fine-tuning on synthetic data created from 3D models, better detection performance on the PASCAL dataset can be achieved than training with only a few labeled examples from the real dataset.

In contrast to our work, most object detection results in computer vision use realistic textures, but do not create coherent 3D scenes. Instead, objects are rendered against a solid background or a randomly chosen photograph. As a result, our approach allows our models to understand the 3D spatial information necessary for rich interactions with the physical world.

Sadeghi and Levine's work [40] is the most similar to our own. The authors demonstrate that a policy mapping images to controls learned in a simulator with varied 3D scenes and textures can be applied successfully to real-world quadrotor flight. However, their experiments – collision avoidance in hallways and open spaces – do not demonstrate the ability to deal with high-precision tasks. Our approach also does not rely on precise camera information or calibration, instead randomizing the position, orientation, and field of view of the camera in the simulator. Whereas their approach chooses textures from a dataset of around 200 pre-generated materials, most of which are realistic, our approach is the first to use only non-realistic textures created by a simple random generation process, which allows us to train on hundreds of thousands (or more) of unique texturizations of the scene.

## III. METHOD

Given some objects of interest $\{s_i\}_i$, our goal is to train an object detector $d(I_0)$ that maps a single monocular camera frame $I_0$ to the Cartesian coordinates $\{(x_i, y_i, z_i)\}_i$ of each object. In addition to the objects of interest, our scenes sometimes contain distractor objects that must be ignored. Our approach is to train a deep neural network in simulation using domain randomization. The remainder of this section describes the specific domain randomization and neural network training methodology we use.

### A. Domain randomization

The purpose of domain randomization is to provide enough simulated variability at training time such that at test time the model is able to generalize to real-world data. We randomize the following aspects of the domain for each sample used during training:

- Number and shape of distractor objects on the table
- Position and texture of all objects on the table
- Textures of the table, floor, skybox, and robot
- Position, orientation, and field of view of the camera
- Number of lights in the scene
- Position, orientation, and specular characteristics of the lights
- Type and amount of random noise added to images

Since we use a single monocular camera image from an uncalibrated camera to estimate object positions, we fix the height of the table in simulation, effectively creating a 2D pose estimation task. Random textures are chosen among the following:

(a) A random RGB value
(b) A gradient between two random RGB values

(c) A checker pattern between two random RGB values

The textures of all objects are chosen uniformly at random – the detector does not have access to the color of the object(s) of interest at training time, only their size and shape. We render images using the MuJoCo Physics Engine's [48] built-in renderer. This renderer is not intended to be photo-realistic, and physically plausible choices of textures and lighting are not needed.

Between $0$ and $10$ distractor objects are added to the table in each scene. Distractor objects on the floor or in the background are unnecessary, despite some clutter (e.g., cables) on the floor in our real images.

Our method avoids calibration and precise placement of the camera in the real world by randomizing characteristics of the cameras used to render images in training. We manually place a camera in the simulated scene that approximately matches the viewpoint and field of view of the real camera. Each training sample places the camera randomly within a $(10 \times 5 \times 10)$ cm box around this initial point. The viewing angle of the camera is calculated analytically to point at a fixed point on the table, and then offset by up to $5.7$ degrees ($0.1$ radians) in each direction. The field of view is also scaled by up to $5\,\%$ from the starting point.

### B. Model architecture and training

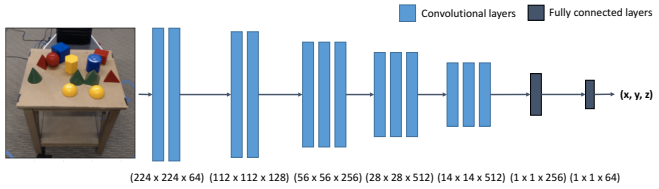


Fig. 2. The model architecture used in our experiments. Each vertical bar corresponds to a layer of the model. ReLU nonlinearities are used throughout, and max pooling occurs between each of the groupings of convolutional layers. The input is an image from an external webcam downsized to $(224 \times 224)$.

We parametrize our object detector with a deep convolutional neural network. In particular, we use a modified version the VGG-16 architecture [42] shown in Figure 2. We chose this architecture because it performs well on a variety of computer vision tasks, and because it has a wide availability of pretrained weights. We use the standard VGG convolutional layers, but use smaller fully connected layers of sizes $256$ and $64$ and do not use dropout. For the majority of our experiments, we use weights obtained by pretraining on ImageNet to initialize the convolutional layers, which we hypothesized would be essential to achieving transfer. In practice, we found that using random weight initialization works as well in most cases.

We train the detector through stochastic gradient descent on the $L_2$ loss between the object positions estimated by the network and the true object positions using the Adam optimizer [20]. We found that using a learning rate of around $1\mathrm{e}{-4}$ (as opposed to the standard $1\mathrm{e}{-3}$ for Adam) improved convergence and helped avoid a common local optimum, mapping all objects to the center of the table.

## IV. EXPERIMENTS

### A. Experimental Setup

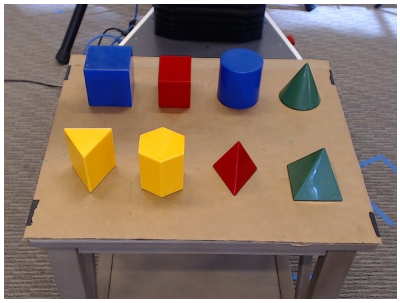

Fig. 3. The geometric objects used in our experiments.

We evaluated our approach by training object detectors for each of eight geometric objects. We constructed mesh representations for each object to render in the simulator. Each training sample consists of (a) a rendered image of the object and one or more distractors (also from among the geometric object set) on a simulated tabletop and (b) a label corresponding to the Cartesian coordinates of the center of mass of the object in the world frame.

For each experiment, we performed a small hyperparameter search, evaluating combinations of two learning rates ($1\mathrm{e}{-4}$ and $2\mathrm{e}{-4}$) and three batch sizes ($25$, $50$, and $100$). We report the performance of the best network.

The goals of our experiments are:

(a) Evaluate the localization accuracy of our trained detectors in the real world, including in the presence of distractor objects and partial occlusions

(b) Assess which elements of our approach are most critical for achieving transfer from simulation to the real world

(c) Determine whether the learned detectors are accurate enough to perform robotic manipulation tasks

TABLE I

| **Detection error for various objects, cm** | | | |
|---|---|---|---|
| Evaluation type | Object only | Distractors | Occlusions |
| Cone | $1.1 \pm 0.4$ | $1.1 \pm 0.3$ | $0.9 \pm 0.4$ |
| Cube | $0.9 \pm 0.5$ | $2.0 \pm 2.2$ | $1.5 \pm 1.1$ |
| Cylinder | $0.9 \pm 0.5$ | $1.9 \pm 3.6$ | $2.6 \pm 3.6$ |
| Hexagonal Prism | $0.7 \pm 0.5$ | $0.6 \pm 0.3$ | $1.0 \pm 1.0$ |
| Pyramid | $0.9 \pm 0.3$ | $1.0 \pm 0.5$ | $1.1 \pm 0.7$ |
| Rectangular Prism | $1.1 \pm 0.4$ | $1.1 \pm 0.3$ | $0.9 \pm 0.4$ |
| Tetrahedron | $0.9 \pm 0.5$ | $1.3 \pm 0.8$ | $3.2 \pm 5.8$ |
| Triangular Prism | $0.9 \pm 0.4$ | $0.9 \pm 0.3$ | $1.6 \pm 3.0$ |

### B. Localization accuracy

To evaluate the accuracy of the learned detectors in the real world, we captured $480$ webcam images of one or more geometric objects on a table at a distance of $70\,\mathrm{cm}$ to $105\,\mathrm{cm}$ from the camera. The camera position remains constant across all images. We did not control for lighting or the rest of the scene around the table (e.g., all images

contain part of the robot and tape and wires on the floor). We measured ground truth positions by aligning the object of interest on a 1 millimeter grid on the tabletop. Note that the grid may add up to 1 mm of error, so we only report up to that resolution. Each of the eight geometric objects has 60 labeled images in the dataset: 20 with the object alone on the table, 20 in which one or more distractor objects are present on the table, and 20 in which the object is partially occluded by another object.

Table I summarizes the performance of our models on the test set. Our object detectors are able to localize objects to within 1.5 cm (on average) in the real world and perform well in the presence of clutter and partial occlusions. Though the accuracy of our trained detectors is promising, note that domain mismatch still causes worse performance than on the simulated training data, where error is around 0.3 cm.

### C. Variability in detector performance

The localization accuracy reported in Table I represents the the performance of the best hyperparameter setting. Table II summarizes how performance of trained detectors varies with different seeds and hyperparameter settings. We used 2 seeds for each of the 6 hyperparemeter settings described in the previous section.

We evaluated each on the test set and a synthetic validation set consisting of the simulated scene with more realistic non-random textures. The first six columns of Table II show that the performance varies significantly between runs on both evaluation sets.

The last column reports the cosine similarity between the performance of all models on the test set and the synthetic validation set. The high similarity scores suggest that relative performance of a model on the synthetic validation set is a good proxy for relative performance on the test set. Choosing the best-performing model on the validation set may be a reasonable strategy for finding the best-performing model in the real world.

TABLE II

| **Detection error across hyperparameters and seeds, cm** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Synthetic validation images | | | Real images | | | Cos |
| | Best | Worst | Avg | Best | Worst | Avg | Sim |
| Cone | 0.3 | 12.5 | $2.3 \pm 3.9$ | 0.9 | 14.3 | $3.4 \pm 4.1$ | 0.98 |
| Cube | 0.2 | 12.7 | $1.5 \pm 3.5$ | 1.5 | 11.4 | $2.9 \pm 2.7$ | 0.91 |
| Cylinder | 0.3 | 12.5 | $1.9 \pm 3.5$ | 1.8 | 14.7 | $3.4 \pm 3.8$ | 0.95 |
| Hex. Prism | 0.4 | 12.6 | $1.9 \pm 3.5$ | 0.8 | 14.8 | $2.5 \pm 3.6$ | 0.98 |
| Pyramid | 0.3 | 12.5 | $1.5 \pm 3.1$ | 1.3 | 14.2 | $3.0 \pm 3.2$ | 0.93 |
| Rect. Prism | 0.5 | 12.8 | $2.1 \pm 3.7$ | 1.0 | 12.5 | $2.6 \pm 3.0$ | 0.96 |
| Tetrahedron | 0.3 | 12.8 | $3.0 \pm 4.7$ | 1.8 | 14.6 | $4.7 \pm 4.7$ | 0.97 |
| Tri. Prism | 0.3 | 12.9 | $2.9 \pm 4.4$ | 1.1 | 14.6 | $4.1 \pm 4.7$ | 0.98 |

In addition to performance variability between seeds and hyperparameters, we looked at the performance variability for the single best detector for each object. The results are summarized in Figures 4 and 5.

Figure 4 summarizes the prediction error on all datapoints in the test set. The mode of each error distribution is similar, suggesting that the difference in performance between objects is largely caused by outliers like not finding the object in the image and predicting the position of the wrong object.

Figure 5 shows how performance varies with the position of the object. Errors are more concentrated toward the outside of the table, but are not consistent across object types.

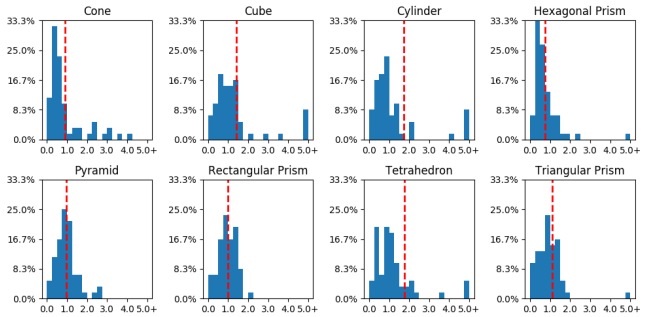


Fig. 4. Error distributions for the best models. The x-axis corresponds to the error (in centimeters), and the y-axis corresponds to the percentage of data points within each bucket. The red line represents the mean performance.

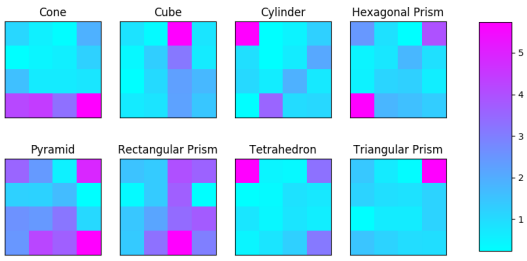


Fig. 5. Error distribution by object position. We divided the table into 4 horizontal and 4 vertical buckets. The bottom-left of each heatmap corresponds to the front-left corner of the table in the camera frame. The scale on the right represents average error in centimeters.

To evaluate the performance of the models on corner cases, we tested create synthetic test images in which (a) the object of interest is not present in the scene, or (b) there are multiple copies of the object of interest on the table. In both cases, the model performed as expected – in the first case, it estimated the position of the object to be the mean of the training images (approximately the center of the table), and in the second case it predicted the object was at the mean position of all the copies of the object of interest.

### D. Performance on non-uniform textures

To examine how our detectors would perform on objects with complex textures, we created simulated scenes in which the object of interest is given a texture from the Describable Textures Dataset (DTD) [4]. Our synthetic test set consists of 7,000 total images using 5 random textures from each of the 35 categories in the DTD. We also gave realistic textures to the table, floor, and background (wood, marble, and a warehouse background respectively). We compared the performance on this dataset to a baseline consisting of scenes with realistically textured table, floor, and background but a uniformly textured object of interest.

The results in Table III show that the model performs comparably when tested on objects with complex, non-uniform textures, suggesting that it has learned a representation invariant to the texture of the object of interest.

TABLE III

| Detection error for various objects, cm | | |
|---|---|---|
| | Complex textures | Baseline |
| Cone | $0.23 \pm 0.09$ | $0.30 \pm 0.14$ |
| Cube | $1.63 \pm 0.37$ | $0.64 \pm 1.37$ |
| Cylinder | $0.30 \pm 0.06$ | $0.34 \pm 0.18$ |
| Hexagonal Prism | $0.46 \pm 0.26$ | $0.37 \pm 0.25$ |
| Pyramid | $0.21 \pm 0.11$ | $0.30 \pm 0.19$ |
| Rectangular Prism | $0.5 \pm 0.26$ | $0.51 \pm 0.69$ |
| Tetrahedron | $0.32 \pm 0.15$ | $0.32 \pm 0.19$ |
| Triangular Prism | $0.27 \pm 0.09$ | $0.29 \pm 0.16$ |

### *E. Comparison to existing methods*

The accuracy of our detectors are comparable at a similar distance to the translation error in traditional techniques for pose estimation in clutter from a single monocular camera frame [6] that use higher-resolution images.

The primary advantage of our technique over existing methods is that it uses a simple-to-implement, scalable pipeline that may be easy to extend to harder problems. However, our technique has several limitations relative to existing approaches. The primary drawback of our method is that it requires setting up simulated scenes and training separate detectors for each object, including creating 3D models for each object. Once trained, the models may not generalize well to new scenes (e.g., new positions of the table relative to the robot or large changes of the position of the camera relative to the table). Finally, since our approach is based on deep neural networks, failure cases can be extreme (e.g., greater than $15\,\mathrm{cm}$ error) and difficult to interpret.

### *F. Ablation study*

To evaluate the importance of different factors of our training methodology, we assessed the sensitivity of the algorithm to the number of training images, the number of unique textures seen in training, the use of random noise in pre-processing, the presence of distractors in training, the randomization of camera position in training, and the use of pre-trained weights in the detection model.

We found that the method is at least somewhat sensitive to all of the factors except the use of random noise.

Figure 6 shows the sensitivity to the number of training samples used for pre-trained models and models trained from scratch. Using a pre-trained model, we are able to achieve relatively accurate real-world detection performance with as few as $5,000$ training samples, but performance improves up to around $50,000$ samples.

Figure 6 shows the performance of a model trained from scratch. Our hypothesis that pre-training would be essential to generalizing to the real world proved to be false. With a large amount of training data, random weight initialization

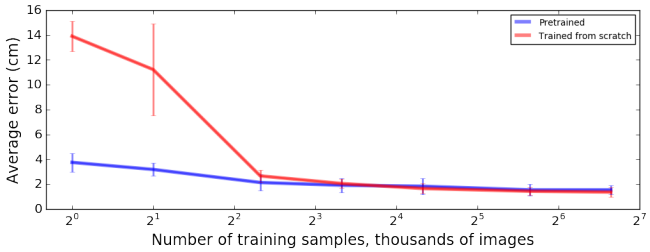


Fig. 6. Sensitivity of test error on real images to the number of simulated training examples used. Each training example corresponds to a single labeled example of an object on the table with between 0 and 10 distractor objects. Lighting and all textures are randomized between iterations.

can achieve nearly the same performance as does pre-trained weight initialization. The best detectors for a given object were often those initialized with random weights. However, using a pre-trained model can significantly improve performance when less training data is used.

Figure 7 shows the sensitivity to the number of unique texturizations of the scene when trained on a fixed number $(10,000)$ of training examples. We found that performance degrades significantly when fewer than $1,000$ textures are used, indicating that for our experiments, using a large number of random textures (in addition to random distractors and object positions) is necessary to achieving transfer. Note that when $1,000$ random textures are used in training, the performance using $10,000$ images is comparable to that of using only $1,000$ images, indicating that in the low data regime, texture randomization is more important than randomization of object positions.

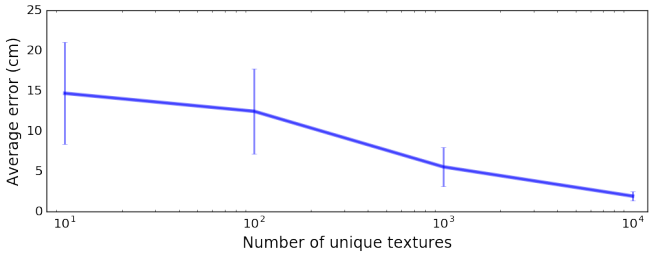


Fig. 7. Sensitivity to amount of texture randomization. In each case, the detector was trained using $10,000$ random object positions and combinations of distractors, but only the given number of unique texturizations and lighting conditions were used.

Table IV examines the performance of the algorithm when random noise, distractors, and camera randomization are removed in training. Incorporating distractors during training appears to be critical to resilience to distractors in the real world. Randomizing the position of the camera also consistently provides a slight accuracy boost, but reasonably high accuracy is achievable without it. Adding noise during pretraining appears to have a negligible effect. In practice, we found that adding a small amount of random noise to images at training time improves convergence and makes training less susceptible to local minima.

TABLE IV

| **Average detection error on geometric shapes by method, cm**[1] | | | |
|---|---|---|---|
| Evaluation type | *Real images* | | |
| | Object only | Distractors | Occlusions |
| Full method | $\mathbf{1.3 \pm 0.6}$ | $\mathbf{1.8 \pm 1.7}$ | $\mathbf{2.4 \pm 3.0}$ |
| No noise added | $1.4 \pm 0.7$ | $1.9 \pm 2.0$ | $\mathbf{2.4 \pm 2.8}$ |
| No camera randomization | $2.0 \pm 2.1$ | $2.4 \pm 2.3$ | $2.9 \pm 3.5$ |
| No distractors in training | $1.5 \pm 0.6$ | $7.2 \pm 4.5$ | $7.4 \pm 5.3$ |

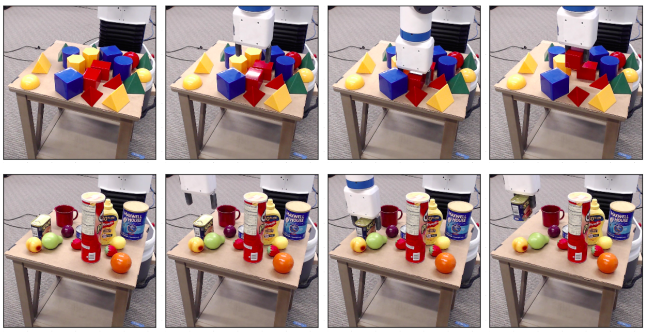

Fig. 8. Two representative executions of grasping objects using vision learned in simulation only. The object detector network estimates the positions of the object of interest, and then a motion planner plans a simple sequence of motions to grasp the object at that location.

### G. Robotics experiments

To demonstrate the potential of this technique for transferring robotic behaviors learned in simulation to the real world, we evaluated the use of our object detection networks for localizing an object in clutter and performing a prescribed grasp. For two of our most consistently accurate detectors, we evaluated the ability to pick up the detected object in 20 increasingly cluttered scenes using the positions estimated by the detector and off-the-shelf motion planning software [44]. To test the robustness of our method to discrepancies in object distributions between training and test time, some of our test images contain distractors placed at orientations not seen during training.

We deployed the pipeline on a Fetch robot [52], and found it was able to successfully pick up the target object in 38 out of 40 trials, including in highly cluttered scenes with significant occlusion of the target object. Note that the trained detectors have no prior information about the color of the target object, only its shape and size, and are able to detect objects placed closely to other objects of the same color.

To test the performance of our object detectors on objects with non-uniform textures, we trained a detector to localize a can of Spam from the YCB Dataset [3]. At test time, instead of using geometric object distractors like in training, we placed other food items from the YCB set on the table. The detector was able to ignore the previously unseen distractors and pick up the target in 9 of 10 trials.

Figure 8 shows examples of the robot grasping trials. For videos, please visit the web page associated with this paper.[2]

[1]Each of the models was trained with $20,000$ training examples

[2]https://sites.google.com/view/domainrandomization/

## V. CONCLUSION

We demonstrated that an object detector trained only in simulation can achieve high enough accuracy in the real world to perform grasping in clutter. Future work will explore how to make this technique reliable and effective enough to perform tasks that require contact-rich manipulation or higher precision.

Future directions that could improve the accuracy of object detectors trained using domain randomization include using higher resolution camera frames, optimizing model architecture choice, introducing additional forms of texture, lighting, and rendering randomization to the simulation, training on more data, incorporating multiple camera viewpoints, stereo vision, or depth information, and combining domain randomization with domain adaptation.

Domain randomization is a promising research direction toward bridging the reality gap for robotic behaviors learned in simulation. Deep reinforcement learning may allow more complex policies to be learned in simulation through large-scale exploration and optimization, and domain randomization could be an important tool for making such policies useful on real robots.

## REFERENCES


[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.

[2] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.

[3] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *Advanced Robotics (ICAR), 2015 International Conference on*, pages 510–517. IEEE, 2015.

[4] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[5] Alvaro Collet, Dmitry Berenson, Siddhartha S Srinivasa, and Dave Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 48–55. IEEE, 2009.

[6] Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research*, 30(10):1284–1306, 2011.

[7] Alvaro Collet and Siddhartha S Srinivasa. Efficient multi-view object recognition and full pose estimation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2050–2055. IEEE, 2010.

[8] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *arXiv preprint arXiv:1407.3501*, 2014.

[9] Mark Cutler and Jonathan P How. Efficient reinforcement learning for robots using informative simulated priors. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2605–2612. IEEE, 2015.

[10] Mark Cutler, Thomas J Walsh, and Jonathan P How. Reinforcement learning with multi-fidelity simulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3888–3895. IEEE, 2014.

[11] Lixin Duan, Dong Xu, and Ivor Tsang. Learning with augmented features for heterogeneous domain adaptation. *arXiv preprint arXiv:1206.4660*, 2012.

[12] Staffan Ekvall, Danica Kragic, and Frank Hoffmann. Object recognition and pose estimation using color cooccurrence histograms and geometric modeling. *Image and Vision Computing*, 23(11):943–955, 2005.

[13] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. *arXiv preprint arXiv:1703.00727*, 2017.

[14] Iryna Gordon and David G Lowe. What and where: 3d object recognition with accurate pose. In *Toward category-level object recognition*, pages 67–82. Springer, 2006.

[15] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *ICLR 2017, to appear*, 2017.

[16] Judy Hoffman, Sergio Guadarrama, Eric Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. Lsda: Large scale detection through adaptation. In *Neural Information Processing Symposium (NIPS)*, 2014.

[17] Judy Hoffman, Erik Rodner, Jeff Donahue, Trevor Darrell, and Kate Saenko. Efficient learning of domain-invariant image representations. *arXiv preprint arXiv:1301.3224*, 2013.

[18] Nick Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, 6(2):325–368, 1997.

[19] Stephen James and Edward Johns. 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*, 2016.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] J Zico Kolter and Andrew Y Ng. Learning omnidirectional path following using dimensionality reduction. In *Robotics: Science and Systems*, 2007.

[22] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2013.

[23] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1785–1792. IEEE, 2011.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[25] Jurgen Leitner, Simon Harding, Mikhail Frank, Alexander Forster, and Jurgen Schmidhuber. Artificial neural networks for spatial perception: Towards visual object localisation in humanoid robots. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE, 2013.

[26] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[27] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779*, 2016.

[28] Chaitanya Mitash, Kostas E Bekris, and Abdeslam Boularias. A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. *arXiv preprint arXiv:1703.03347*, 2017.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[30] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5307–5314. IEEE, 2015.

[31] Yair Movshovitz-Attias, Takeo Kanade, and Yaser Sheikh. How useful is photo-realistic rendering for visual learning? In *Computer Vision–ECCV 2016 Workshops*, pages 202–217. Springer, 2016.

[32] Ramakant Nevatia and Thomas O Binford. Description and recognition of curved objects. *Artificial Intelligence*, 8(1):77–98, 1977.

[33] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

[34] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286, 2015.

[35] Benjamin Planche, Ziyan Wu, Kai Ma, Shanhui Sun, Stefan Kluckner, Terrence Chen, Andreas Hutter, Sergey Zakharov, Harald Kosch, and Jan Ernst. Depthsynth: Real-time realistic synthetic data generation from cad models for 2.5 d recognition. *arXiv preprint arXiv:1702.08558*, 2017.

[36] Aravind Rajeswaran, Sarvjeet Ghotra, Sergey Levine, and Balaraman Ravindran. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

[37] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer, 2016.

[38] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[39] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[40] Fereshteh Sadeghi and Sergey Levine. (cad) ?2 rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[41] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[42] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[43] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2686–2694, 2015.

[44] Ioan A Sucan and Sachin Chitta. Moveit! *http://moveit.ros.org*.

[45] Baochen Sun and Kate Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. In *BMVC*, volume 1, page 3, 2014.

[46] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[47] Jie Tang, Stephen Miller, Arjun Singh, and Pieter Abbeel. A textured object recognition pipeline for color and depth image data. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3467–3474. IEEE, 2012.

[48] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[49] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting deep visuomotor representations with weak pairwise constraints. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[50] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[51] Jur Van Den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu, Andrew Wan, Xiao-Yu Fu, Ken Goldberg, and Pieter Abbeel. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2074–2081. IEEE, 2010.

[52] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. Fetch and freight: Standard platforms for service robot applications. In *Workshop on Autonomous Mobile Service Robots*, 2016.

[53] Patrick Wunsch and Gerd Hirzinger. Real-time visual tracking of 3d objects with dynamic handling of occlusion. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 2868–2873. IEEE, 1997.

[54] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[55] Wenhao Yu, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[56] Fangyi Zhang, Jürgen Leitner, Ben Upcroft, and Peter I. Corke. Vision-based reaching using modular deep networks: from simulation to the real world. *CoRR*, abs/1610.06781, 2016.