Transporter Networks: Rearranging the Visual World for Robotic Manipulation

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien
Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong
Vikas Sindhwani, Johnny Lee
Robotics at Google

Abstract. Robotic manipulation can be formulated as inducing a sequence of spatial displacements: where the space being moved can encompass an object, part of an object, or end effector. In this work, we propose the Transporter Network, a simple model architecture that rearranges deep features to infer spatial displacements from visual input – which can parameterize robot actions. It makes no assumptions of objectness (e.g. canonical poses, models, or keypoints), it exploits spatial symmetries, and is orders of magnitude more sample efficient than our benchmarked alternatives in learning vision-based manipulation tasks: from stacking a pyramid of blocks, to assembling kits with unseen objects; from manipulating deformable ropes, to pushing piles of small objects with closed-loop feedback. Our method can represent complex multi-modal policy distributions and generalizes to multi-step sequential tasks, as well as 6DoF pick-and-place. Experiments on 10 simulated tasks show that it learns faster and generalizes better than a variety of end-to-end baselines, including policies that use ground-truth object poses. We validate our methods with hardware in the real world. Experiment videos and code will be made available at https://transporternets.github.io

Keywords: Deep Learning, Vision, Manipulation

1 Introduction

End-to-end models that map directly from pixels to actions hold the capacity to learn complex manipulation skills, but are known to require copious amounts of data [1, 2]. Integrating object-centric assumptions – e.g., object keypoints [3, 4, 5, 6], embeddings [7, 8], or dense descriptors [9, 10, 11] – has been shown to improve sample efficiency [10]. However, these representations often impose data collection burdens (i.e., configuring scenes with specific singulated objects) and still struggle to address challenging scenarios with unseen classes of objects, occluded objects, highly deformable objects, or piles of small objects [12]. This naturally leads us to ask: is there structure that we can incorporate into our end-to-end models to improve their learning efficiency, without imposing any of the limitations or burdens of explicit object representations?

In this work, we propose the Transporter Network, a simple end-to-end model architecture that preserves *spatial structure* for vision-based manipulation, without object-centric assumptions:

- Manipulation involves rearranging things, which can be thought of as executing a sequence of spatial displacements: where the space being moved (i.e., transported) can encompass an object(s), part of an object, or end effector. We formulate vision for manipulation as estimating these displacements. Transporter Networks directly optimize for this by learning to 1) attend to a local region, and 2) predict its target spatial displacement via deep feature template matching – which then parameterizes robot actions for manipulation. This formulation enables high-level perceptual reasoning about which visual cues are important, and how they should be rearranged in a scene – the distributions of which can be learned from demonstrations.

- Transporter Networks preserve the 3D spatial structure of the visual input. Prior end-to-end models [1, 2] often use convolutional architectures with raw images, in which valuable spatial information can be lost to perspective distortions. Our method uses 3D reconstruction to project visual data onto a spatially-consistent representation as input, with which it is able to better exploit equivariance [13, 14] for inductive biases that are present within the geometric symmetries [15] of the data for more efficient learning.

In our experiments, Transporter Networks exhibit superior sample efficiency on a number of tabletop manipulation tasks that involve changing the state of the robot's environment in a purposeful manner: from stacking a pyramid of blocks, to assembling kits with unseen objects; from manipulating deformable ropes, to pushing piles of small objects with closed-loop feedback. Transporter Networks excel in modeling
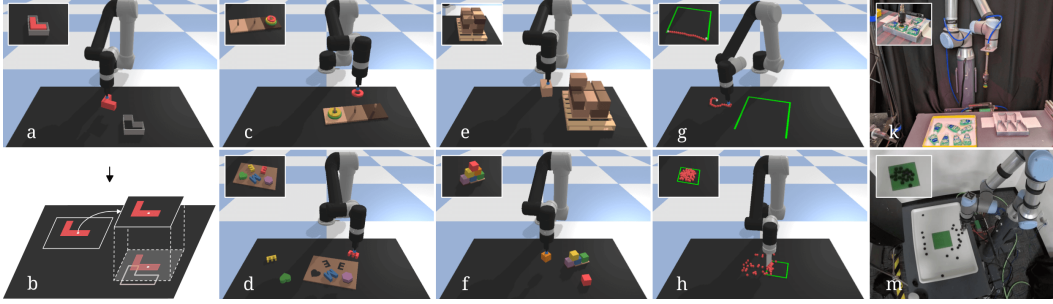
Figure 1. A Transporter Network is a simple model architecture that attends to a local region and predicts its spatial displacement (b) from visual input – which can parameterize robot actions. It is sample efficient in learning complex vision-based manipulation tasks: inserting blocks into fixtures (a), sequential pick-and-place in Towers of Hanoi (c), assembling kits with unseen objects (d), palletizing boxes (e), stacking a pyramid of blocks (f), manipulating rope (g), and pushing piles of small objects with closed-loop feedback (h) – and is practical to deploy on real production robots (k, m).

multi-modal spatial action distributions, and by construction generalize across rotations and translations of objects. They do not require any prior knowledge of the objects to be manipulated – they rely only on information contained within partial RGB-D data from demonstrations, and are capable of generalizing to new objects and configurations, and for some tasks, one-shot learning from a single demonstration.

Our main contribution is a new perspective on the role of spatial structure and its capacity to improve end-to-end learning for vision-based manipulation. We propose a simple model architecture that learns to attend to a local region and predict its spatial displacement, while retaining the spatial structure of the visual input. On 10 unique tabletop manipulation tasks, Transporter Networks trained from scratch are capable of achieving greater than 90% success on most tasks with objects in new configurations using 100 expert demonstrations, while other end-to-end alternatives struggle to generalize with the same amount of data. We also develop an extension to 6DoF tasks by combining 3DoF Transporter Networks with continuous regression to handle the remaining degrees of freedom. To facilitate further research in vision-based manipulation, we plan release code and open-source Ravens, our new simulated benchmark with all tasks. Ravens features a Gym-like API [16] with a built-in stochastic oracle to evaluate the sample efficiency of imitation learning methods.

2 Related Work

Object-centric Representations. Considerable research in vision for manipulation has been devoted to object detectors and pose estimators [17, 18, 19, 20, 21]. These methods often require object-specific training data, making them difficult to scale to applications with unseen objects. Alternative representations including keypoints [3, 4, 5, 6] and dense descriptors [9, 10, 11] have shown to be capable of category-level generalization and representing deformable objects, but still struggle to represent scenes with an unknown number of objects (e.g., piles of small objects) or occluded objects. End-to-end models for vision-based manipulation learn faster when integrated with such object-centric representations [10], providing sample-efficiency benefits but retaining the data-collection and representational constraints of object-centricness. In this work, we show that it is possible to achieve sample efficient end-to-end learning without object-centric representations – enabling the same model architecture to generalize to tasks with unseen objects, variable numbers of objects, deformable objects, and piles of small objects.

Pick-and-Place has had a long history in robotics, motivated by industrial needs. Classic systems use pose estimation of known objects [17, 18], as well as scripted planning and motion control [22]. While robust in structured settings (e.g., manufacturing), these systems are difficult to deploy in unstructured settings (e.g., logistics, agriculture, household) – leading to renewed interest in leveraging learned models to achieve general pick-and-place policies [23, 24, 25, 26, 27, 28, 29, 30] that can handle unseen objects. We benchmark against the more recent Form2Fit [24] on multiple tasks, and demonstrate that Transporter Networks are better able to handle tasks that require precise placing, multi-step sequencing, and closed-loop visual feedback.

3 Method

Consider the problem of learning pick-and-place actions $\mathbf{a}_t$ with a robot from visual observations $\mathbf{o}_t$:

$$f(\mathbf{o}_t) \rightarrow \mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}) \in \mathcal{A}$$

where $\mathcal{T}_{\text{pick}}$ is the pose of the end effector used to pick an object, and $\mathcal{T}_{\text{place}}$ to place the object. Both poses can be defined in SE(2) or SE(3), depending on the task and degrees of freedom available to the space of the end-effector

2

end effector. We can generalize beyond pick-and-place by considering manipulation with other motion primitives [22] that are also parameterized by two end effector poses. For example, we can parameterize pushing, where $\mathcal{T}_{\text{pick}}$ is instead the pose of the end effector at the start of the push, and $\mathcal{T}_{\text{place}}$ is where it moves to complete the push. In this work, we consider tasks that can be completed by a sequence of two-pose motion primitives. In the following section, we begin by describing our approach in the context of planar pick-and-place. We then discuss extensions to 6DoF pick-and-place, sequential multi-step tasks, and motion-primitive-based manipulation of deformable objects and piles of objects.

3.1 Learning to Transport

Consider the planar pick-and-place problem on the right, where $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}} \in \mathbb{R}^2$ are 2D coordinates. In this task, the goal is to pick up the red block with an immobilizing grasp (e.g., with a suction gripper), then place it into the fixture. This simple setting reveals two fundamental aspects of the pick-and-place problem: (i) there may be a distribution of successful pick poses (from which $\mathcal{T}_{\text{pick}}$ may be sampled e.g., on the surface of the block), and (ii) for each successful pick pose, there is a corresponding distribution of successful place poses (from which $\mathcal{T}_{\text{place}}$ may be sampled).
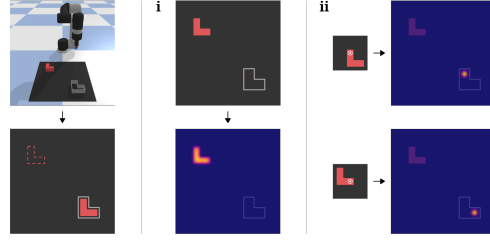

Figure 2. Simple planar pick-and-place task where (i) there is a distribution of successful pick poses, and (ii) for each successful pick pose, there is a corresponding distribution of successful place poses.

The goal of Transporter Networks is to recover these distributions from visual observations alone – without assumptions of objectness. While assigning canonical poses to the block and fixture might simplify the problem, there are many benefits to avoiding such object-centric representations – including the capacity to handle unseen objects, deformable objects, and uncountable piles of small objects. Hence we assume no prior information about the objects (e.g., no 3D models, poses, class categories, keypoints, etc.).

Transporter Networks decompose the problem into (i) picking and (ii) pick-conditioned placing:

$$f_{\text{pick}}(\mathbf{o}_t) \rightarrow \mathcal{T}_{\text{pick}} \qquad\qquad f_{\text{place}}(\mathbf{o}_t, \mathcal{T}_{\text{pick}}) \rightarrow \mathcal{T}_{\text{place}}$$

We present a solution to both with Transporter Networks – but in this work, our primary contribution is on (ii), pick-conditioned placing via transporting. We first begin by briefly describing (i), picking.

Learning Picking. Our visual observation $\mathbf{o}_t$ is a projection of the scene (e.g., reconstructed from RGB-D images), defined on a regular grid of pixels $\{(u,v)\}$ at timestep t of a sequential rearrangement task. Through camera-to-robot calibration, we can correspond each pixel in $\mathbf{o}_t$ to a picking action at that location: $\mathcal{T}_{\text{pick}} \sim (u,v) \in \mathbf{o}_t$. The distribution of successful picks over pixels in $\mathbf{o}_t$ can be multi-modal in nature – particularly in cases when there are multiple instances of the same object in the scene, or when there are symmetries in the shapes of objects. As in prior works [31, 32, 33], Transporter Networks use fully convolutional networks (FCNs, commonly used in vision for image segmentation tasks [34]) to model an action-value function $\mathcal{Q}_{\text{pick}}((u,v)|\mathbf{o}_t)$ that correlates with picking success (architecture in Sec. 3.2):

$$\mathcal{T}_{\text{pick}} = \underset{(u,v)}{\arg\max}\ \mathcal{Q}_{\text{pick}}((u,v)|\mathbf{o}_t)$$

FCNs are translationally equivariant by nature, which synergizes with spatial action representations: if an object to be picked in the scene is translated, then the picking pose also translates. Formally, equivariance here can be characterized as $f_{\text{pick}}(g \circ \mathbf{o}_t) = g \circ f_{\text{pick}}(\mathbf{o}_t)$, where g is a translation. Spatial equivariance has previously been shown to improve learning efficiency for vision-based picking [29, 31, 32, 33, 35]. To this end, FCNs excel in modeling complex multi-modal picking distributions anchored on visual features.

Spatially Consistent Visual Representations. The benefits of spatial equivariance are most pronounced when observations $\mathbf{o}_t$ are spatially consistent, meaning that the appearance of an object remains constant across different camera views, as well as across spatial transforms of rigid objects. This is in contrast to perspective images, in which the appearance of an object could scale or distort depending on the camera view and lens properties. In practice, spatial consistency also simplifies data augmentation during training, since applying rigid transforms to $\mathbf{o}_t$ can yield data that appears as if objects in the scene were physically reconfigured. As with perspective images, however, spatially consistent representations still retain challenges from partial visibility and occlusion. As in [33], we convert RGB-D images into a spatially consistent form by unprojecting to a 3D point cloud, then rendering into an *orthographic* projection, in which each pixel (u,v) represents a fixed window of 3D space (Fig. 3a). Unlike in prior work, we use this spatially consistent representation for the *transport* operation, which we describe next.
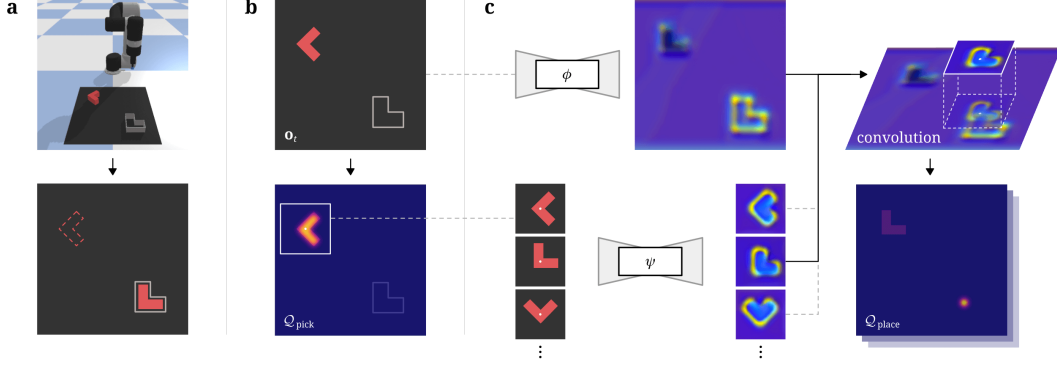
Figure 3. In this setting (a) where the task is to pick up the red block with an immobilizing grasp (e.g., suction) and place it into the fixture, the goal of Transporter Networks is to recover the distribution of successful picks (b), and distribution of successful placements (c) conditioned on a sampled pick. For pick-conditioned placing (c), deep feature template matching occurs with a local crop around the sampled pick as the exemplar. Rotations of the crop around the pick are used to decode the best placing rotation. Our method preserves rotation and translation equivariance for efficient learning.

Learning Pick-Conditioned Placing via Transporting. Spatially consistent visual representations enable us to perform visuo-spatial *transporting*, in which dense pixel-wise features from a partial crop $\mathbf{o}_t[\mathcal{T}_{\text{pick}}]$ centered on $\mathcal{T}_{\text{pick}}$ are rigidly transformed then overlaid on top of another partial crop $\mathbf{o}_t[\tau]$ centered on a candidate place pose τ, where $\mathbf{o}_t$ is the observation before the pick. Intuitively, if the crop encompasses an object, then this operation can be thought of as imagining the object at another location in the scene. Fig. 3c shows this in the context of overlaying the red block across the scene to find its placement in the fixture.

In this work, the goal of Transporter Networks is to transport our partial crop $\mathbf{o}_t[\mathcal{T}_{\text{pick}}]$ densely across a set of poses $\{\tau_i\}$ to search for its best placement, i.e. the $\mathbf{o}_t[\tau_i]$ with the highest feature correlation. This operation is analogous to the search function in visual tracking literature [36, 37, 38] where $\mathbf{o}_t[\mathcal{T}_{\text{pick}}]$ is the exemplar and $\mathbf{o}_t$ is the search area, or in retrieval systems literature where $\mathbf{o}_t[\mathcal{T}_{\text{pick}}]$ is the query and $\mathbf{o}_t$ is the key. We formulate this as a template matching problem, using cross-correlation with dense feature embeddings $\psi(\cdot)$ and $\phi(\cdot)$ from two deep models:

$$\mathcal{Q}_{\text{place}}(\tau|\mathbf{o}_t,\mathcal{T}_{\text{pick}})=\psi(\mathbf{o}_t[\mathcal{T}_{\text{pick}}])*\phi(\mathbf{o}_t)[\tau], \qquad \mathcal{T}_{\text{place}}=\underset{\{\tau_i\}}{\arg\max}\ \mathcal{Q}_{\text{place}}(\tau_i|\mathbf{o}_t,\mathcal{T}_{\text{pick}}) \tag{1}$$

where $\mathcal{Q}_{\text{place}}(\tau|\mathbf{o}_t,\mathcal{T}_{\text{pick}})$ is an action-value function that correlates with placing success, and τ covers the space of all possible placing poses – which is equivalent to the space of 2D translations across $\mathbf{o}_t$ in our first simple red block and fixture example. Since $\mathbf{o}_t$ is spatially consistent, the structure of dense features $\psi(\mathbf{o}_t)$ and $\phi(\mathbf{o}_t)$ are also spatially consistent. A key property is that Eq. 1 is *invariant* to $\mathcal{T}_{\text{pick}}$ as long as cross-correlation via transform $\mathcal{T}_{\text{place}}$ produces the desired imagined overlay. This enables learning pick-conditioned placing from few examples (Fig. 5), without having to train over all possible picks [27]. Note that it is the deep features, not the RGB-D pixels, which are overlaid, and that the influence of the overlay extends beyond the crop, since for each dense pixel in feature space, the receptive field is large.

Learning with Planar Rotations in SE(2). Our first example uses only pick-and-place actions in the space of 2D translations – however, $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$ may also represent rotations as in Fig. 3. In the planar SE(2) case, we discretize the space of SO(2) rotations into k bins, then rotate the input visual observation $\mathbf{o}_t$ for each bin. $\mathbf{o}_t$ is thereby defined on a grid of voxels $\{(u,v,w)_i\}$ where w lies on the k-discretized axis of rotations, and $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$ are represented discretely on this grid in SE(2). In practice, this is achieved by running the FCN k times in parallel, once for each rotated $\mathbf{o}_t$, sharing model weights across all rotations.

Learning with Full Rigid Body Transformations in SE(3). While the previous sections discussed learning a $\mathcal{Q}$ function over a discretized space, extending to the full SE(3) case to handle all six degrees of freedom of rigid objects becomes infeasible to sufficiently discretize the 6-dimensional space – due to the curse of dimensionality. However, interestingly we find the transport operation to be an enabling building block into higher dimensions, using a multi-stage approach to extend into

Figure 4. In this 6-DoF variant of the task presented in Fig. 3, the fixture location varies as well in the out-of-plane rotations (r_x, r_y), and height (z). Transporter Networks can address this task through multi-stage inference.

tasks which require SE(3) placing. First, the three SE(2) degrees-of-freedom are addressed as in the previous section, producing an estimated $\hat{\mathcal{T}}_{\text{place}}^{\text{SE(2)}}$. Then, we extend the cross-correlation transport operation
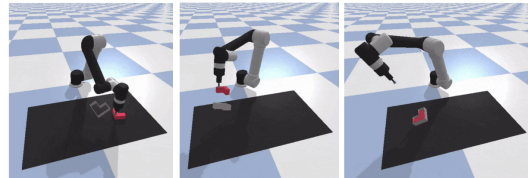
4

– rather than using one channel over SE(2) to classify the correct discretized action, we use three channels over SE(2) to regress the remaining rotational (r_x, r_y) and translational (z-height) degrees of freedom. We use a ψ' and ϕ' which are identical to the ψ and ϕ used for SE(2) classification, but with three separate cross-correlations ($*_3$) from subsets of channels, and also add learnable nonlinearity after the cross-correlation via a three-headed MLP network, $f(\cdot)$: $r_x, r_y, z = f\big(\psi'(\mathbf{o}_t[\mathcal{T}_{\text{pick}}]) *_3 \phi'(\mathbf{o}_t)[\hat{\mathcal{T}}_{\text{place}}^{\text{SE(2)}}]\big)$. This hybrid discrete/continuous approach affords a number of nice properties: the discrete SE(2) step is good at representing complex multi-modal distributions in those degrees of freedom, and once an approximate SE(2) alignment has been *overlaid* via the transport operation, this acts as an attention mechanism to aid the model in precise continuous regression. To handle multi-modality in the continuous degrees of freedom, we can regress mixture densities. In practice we have found that multiple approaches can work well once approximate SE(2) alignment has been applied, see the Appendix for more. Multi-stage SE(3) classification, using Eq. 1 iteratively, is also possible.

Learning Sequential Pick-and-Place. Consider the challenging example with 3-disk Towers of Hanoi in Fig. 1c, where the task is to sequentially move 3 disks from the first tower to the third without placing a larger disk over a smaller disk. Completing this task requires correctly sequencing 7 pick-and-place actions. In this setup with a unique coloring of the base and disks, the task is Markovian. Hence actions can be directly conditioned on visual contextual cues, e.g., the next pick-and-place depends on the position (or absence due to occlusion) of observed objects in the scene. Our method is stateless, but experiments show that it can learn such sequencing behaviors through visual feedback. To enable this, it is important to increase the receptive field [39] of the FCNs to encompass most of the visual observation $\mathbf{o}_t$. Extending Transporter Networks with memory to handle non-Markovian tasks would be interesting future work.

Manipulation Beyond Pick-and-Place: Deformables and Piles of Objects. Beyond pick-and-place of rigid objects, Transporter Networks can generalize to tasks that can be completed with two-pose primitives. For example, they can learn to sequentially rearrange a deformable rope such that it connects the two end points of an incomplete 3-sided square (Fig. 1e), or sequentially push piles of small objects into a desired target set (Fig. 1h) using a spatula-shaped end effector. In both cases, our experiments (Sec. 4) demonstrate that they can learn closed-loop behaviors using visual feedback to complete these tasks. Both tasks are characterized by dynamics that cannot be modeled with rigid displacements – however, our results suggest that rigid displacements can serve as a useful prior for non-rigid ones.

3.2 Network Architecture

Observation Space: For tabletop manipulation, our visual observation $\mathbf{o}_t$ is an orthographic top-down view of a 0.5×1m tabletop workspace, generated in simulation by fusing 480×640 RGB-D images captured with calibrated cameras using known intrinsics and extrinsics. The top-down image $\mathbf{o}_t$ has a pixel resolution of 160×320 – each pixel represents a 3.125×3.125mm vertical column of 3D space in the workspace. The image $\mathbf{o}_t$ contains both color (RGB) and scalar height-from-bottom (H) information, which enables the deep models to learn features that are rich in both visual texture and geometric shape.

Picking: Our picking model is a single feed-forward FCN that takes as input the visual observation $\mathbf{o}_t \in \mathbb{R}^{H \times W \times 4}$ and outputs dense pixel-wise values that correlate with picking success: $\mathcal{V}_{\text{pick}} \in \mathbb{R}^{H \times W} = \text{softmax}(\mathcal{Q}_{\text{pick}}((u,v)|\mathbf{o}_t))$. Our picking model is an hourglass encoder decoder architecture: a 43-layer residual network (ResNet) [40] with 12 residual blocks and 8-stride (3 2-stride convolutions in the encoder and 3 bilinear-upsampling layers in the decoder), followed by image-wide softmax. Each convolutional layer after the first is equipped with dilation [41], and interleaved with ReLU activations [42] before the last layer. 8-stride was chosen to balance between maximizing receptive field coverage per pixel prediction, while minimizing loss of resolution in the latent mid-level features of the network.

Placing: Our placing model is a two-stream feed-forward FCN that takes as input the visual observation $\mathbf{o}_t \in \mathbb{R}^{H \times W \times 4}$ and outputs two dense feature maps: query features $\psi(\mathbf{o}_t) \in \mathbb{R}^{H \times W \times d}$ and key features $\phi(\mathbf{o}_t) \in \mathbb{R}^{H \times W \times d}$, where d is the feature dimensionality. Our placing model shares a similar hourglass encoder decoder architecture as the picking model: each stream is an 8-stride 43-layer ResNet, but without non-linear activations in the last layer. A partial crop $\psi(\mathbf{o}_t[\mathcal{T}_{\text{pick}}]) \in \mathbb{R}^{c \times c \times d}$ of query features with size c centered around $\mathcal{T}_{\text{pick}} = \text{argmax}\,\mathcal{V}_{\text{pick}}$ is transformed by τ (which covers the space of possible placement configurations), then cross-correlated (Eq. 1) with the key feature map $\phi(\mathbf{o}_t)$ to output dense pixel-wise values that correlate with placing success: $\mathcal{V}_{\text{place}} \in \mathbb{R}^{H \times W \times k} = \text{softmax}(\mathcal{Q}_{\text{place}}(\tau|\mathbf{o}_t, \mathcal{T}_{\text{pick}}))$ where τ is discretized into $k=36$ angles (multiples of $10°$). The translation and rotation τ which gives the highest correlation with the partial crop, via $\psi(\mathbf{o}_t[\mathcal{T}_{\text{pick}}]) * \phi(\mathbf{o}_t)[\tau]$, gives $\mathcal{T}_{\text{place}} = \text{argmax}\,\mathcal{V}_{\text{place}}$. Note that this operation can be made fast if implemented with highly optimized matrix multiplication (where the crop is

flattened) or as a convolution (where the crop is the convolving kernel). Total inference time (with both picking and placing models) amounts to 200ms on an Nvidia GTX 2080 GPU.

3.3 Training: Learning from Demonstrations

We assume access to a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, ..., \zeta_n\}$ of n expert demonstrations, where each trajectory $\zeta_i = \{(\mathbf{o}_0, \mathbf{a}_0), (\mathbf{o}_1, \mathbf{a}_1), ...\}$ is a sequence of one or more observation-action pairs $(\mathbf{o}_t, \mathbf{a}_t)$. In our experiments (Sec. 4), we evaluate Transporter Networks in their ability to learn from $n = 1, 10, 100$, or 1,000 demonstrations per task. Our method does not take any task-specific information as input, and is trained on one task per model (future work may investigate multi-task). During training, we uniformly sample observation-action pairs from the dataset $\mathcal{D}$, from which each action $\mathbf{a}_t$ can be unpacked into two training labels: $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$, used to generate binary one-hot pixel maps $Y_{\text{pick}} \in \mathbb{R}^{H \times W}$ and $Y_{\text{place}} \in \mathbb{R}^{H \times W \times k}$ respectively. Our training loss is simply the cross-entropy between these one-hot pixel maps and the outputs of the picking and placing models: $\mathcal{L} = -\mathbb{E}_{Y_{\text{pick}}}[\log \mathcal{V}_{\text{pick}}] - \mathbb{E}_{Y_{\text{place}}}[\log \mathcal{V}_{\text{place}}]$. While we only have labels for a single pixel per dense probability map, gradients are passed to all other pixels via image-wide softmax. We find this loss to be capable of learning multi-modal non-isotropic spatial action distributions. For SE(3) models involving regression, we use a Huber loss on each regression channel. Validation performance generally converges after a few hours of training on a single commodity GPU.

4 Results

We execute experiments in both simulated and real settings to evaluate Transporter Networks across various tasks. Our goals are threefold: 1) investigate whether preserving visuo-spatial structure within Transporter Networks improves their sample efficiency and generalization, 2) compare them to common baselines for end-to-end vision-based manipulation, and 3) demonstrate that they are practical to deploy on real robots.

4.1 Simulation Experiments

We use behavior cloning simulation experiments to compare with baselines. In this work, we do not use simulation for sim-to-real transfer – rather only as a means to provide a consistent and controlled environment for fair comparisons and ablations. Ravens, our simulated benchmark learning environment built with PyBullet [43], consists of a Universal Robot UR5e with a suction gripper overlooking a 0.5×1m tabletop workspace, with 3 simulated 640x480 RGB-D cameras pointing towards the workspace: one stationed in front (facing the robot), one on the left shoulder of the robot arm, and one on the right shoulder. Our method does not require 3 cameras – having more cameras only improves visual coverage. For example, our real experiments only use 1 camera per robot.

Task	precise placing	multimodal placing	multi-step sequencing	unseen objects
block-insertion (Fig. 3)	✓	✗	✗	✗
place-red-in-green	✗	✓	✗	✗
towers-of-hanoi (Fig. 7)	✓	✓	✓	✗
align-box-corner	✓	✓	✗	✓
stack-block-pyramid[*]	✓	✓	✓	✗
palletizing-boxes[*§]	✓	✓	✓	✗
assembling-kits[*§] [24]	✓	✓	✗	✓
packing-boxes[*§]	✓	✓	✓	✓
manipulating-rope[*†]	✓	✓	✓	✗
sweeping-piles[*†] [12]	✓	✓	✓	✗

[*]tasks that have more than one correct sequence of states.
[†]tasks that are beyond pick-and-place of rigid objects.
[§]tasks that are commonly found in industry.

Table 1. Each task in Ravens is characterized by a unique set of attributes.

Tasks and Metrics. In simulation, we benchmark on 10 discrete-time tabletop manipulation tasks, some which require closed-loop visual feedback for multi-step sequencing. We position-control the robot end effector (suction cup, or spatula for pushing) with two-pose motion primitives [22]. For each task, all objects (including target zones) are randomly positioned and oriented in the workspace during training and testing. Each task comes with a scripted oracle that provides expert demonstrations by randomly sampling the distribution of successful $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$ – samples of these are made available for the learner. Some tasks include multi-modality and permutations – for example, in stacking, a successful 6-block pyramid is invariant to the permutation of blocks within each row, but not between rows. Performance is evaluated with a metric from 0 (failure) to 100 (success). Partial credit is assigned during tasks that require sequencing multiple actions. We report results with the models that have achieved highest validation performance during training, averaged over 100 unseen test runs for each task. More details in the Appendix.

Baseline Methods. *Form2Fit* [24], as mentioned in Related Work, is different from our method in that it does not perform visuo-spatial transporting, but instead uses a matching network. *ConvMLP* is a common model architecture in the end-to-end literature [1] that uses a series of convolutional layers, followed by

spatial-softmax and a multi-layer perceptron (MLP) – here, we use this to regress the sequence of two SE(2) poses. Form2Fit and ConvMLP use the same input images as Transporter Networks, but to get a sense of the difficulty of our tasks, we also supply two baselines which consume ground truth state (object poses and 3D bounding boxes) as input – i.e., assuming perfect object poses. *GT-State MLP* is an MLP which consumes these and regresses two SE(2) poses, while *2-step GT-State MLP* first regresses the first SE(2) pose, then adds this to the observation vector and regresses the second SE(2) pose . To represent multi-modality, ConvMLP, GT-State MLP, and 2-step GT-State MLP regress mixture densities [44]. All methods, including ours, use identical data augmentation, with random SE(2) world-frame transforms.

Results: Sample Efficiency and Generalization on Benchmark Tasks

Tab. 2 shows sample efficiency of baselines trained from stochastic demonstrations for each task, and evaluated on unseen test settings, with random rotations and translations of objects (including target zones). The benchmark is difficult – most baselines, while capable of over-fitting to the demonstration training set, generalize poorly with only 1,000 demonstrations. In general, Transporter Networks achieve orders of magnitude more sample efficiency than the image-based alternatives, and also provides better sample efficiency than multi-layer perceptrons trained with ground truth state. More analysis in the Appendix.

	block-insertion				place-red-in-green				towers-of-hanoi				align-box-corner				stack-block-pyramid			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter Network	**100**	**100**	**100**	**100**	**84.5**	**100**	**100**	**100**	**73.1**	**83.9**	**97.3**	**98.1**	35.0	**85.0**	**97.0**	**98.3**	13.3	**42.6**	**56.2**	**78.2**
Form2Fit [24]	17.0	19.0	23.0	29.0	83.4	**100**	**100**	**100**	3.6	4.4	3.7	7.0	7.0	2.0	5.0	16.0	**19.7**	17.5	18.5	32.5
Conv. MLP	0.0	5.0	6.0	8.0	0.0	3.0	25.5	31.3	0.0	1.0	1.9	2.1	0.0	2.0	1.0	1.0	0.0	1.8	1.7	1.7
GT-State MLP	4.0	52.0	96.0	99.0	0.0	0.0	3.0	82.2	10.7	10.7	6.1	5.3	47.0	29.0	29.0	59.0	0.0	0.2	1.3	15.3
GT-State MLP 2-Step	6.0	38.0	95.0	**100**	0.0	0.0	19.0	92.8	22.0	6.4	5.6	3.1	**49.0**	12.0	43.0	55.0	0.0	0.8	12.2	17.5

	palletizing-boxes				assembling-kits				packing-boxes				manipulating-rope				sweeping-piles			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter Network	**63.2**	**77.4**	**91.7**	**97.9**	**28.4**	**78.6**	**90.4**	**94.6**	**56.8**	**58.3**	**72.1**	**81.3**	**21.9**	**73.2**	**85.4**	**92.1**	**52.4**	**74.4**	**71.5**	**96.1**
Form2Fit [24]	21.6	42.0	52.1	65.3	3.4	7.6	24.2	37.6	29.9	52.5	62.3	66.8	11.9	38.8	36.7	47.7	13.2	15.6	26.7	38.4
Conv. MLP	31.4	37.4	34.6	32.0	0.0	0.2	0.2	0.0	0.3	9.5	12.6	16.1	3.7	6.6	3.8	10.8	28.2	48.4	44.9	45.1
GT-State MLP	0.6	6.4	30.2	30.1	0.0	0.0	1.2	11.8	7.1	1.4	33.6	56.0	5.5	11.5	43.6	47.4	7.2	20.6	63.2	74.4
GT-State MLP 2-Step	0.6	9.6	32.8	37.5	0.0	0.0	1.6	4.4	4.0	3.5	43.4	57.1	6.0	8.2	41.5	58.7	9.7	21.4	66.2	73.9

Table 2. **Baseline comparisons.** Task success rate (mean %) vs. # of demonstration episodes (1, 10, 100, or 1000) used in training.

Generalizing to Unseen Objects. Three of our benchmark tasks involve generalizing to unseen objects (see Tab. 1). In our variant of the kit assembly task with unseen objects (shown in Fig. 8), the gap in performance between Form2Fit and Transporter Networks is rather large. On a simpler set of unseen objects that have more primitive geometries (e.g., circular disks, squares), which reflect the distribution of objects used in the original paper [24], Form2Fit achieves 96.3% task success. This suggests that Form2Fit descriptors have the capacity to express larger differences in geometry, but are less capable of matching higher resolution information than our deep template-matching based model, with the same amount of data.

Learning Sequential Manipulation with Closed-Loop Visual Feedback. In this work, Transporter Networks are stateless models that react only to information presented as visual input during the current timestep. However, our experiments show that the models have the capacity to learn visual feedback: they make use of contextual visual cues to determine which step of the task they are in, and use that to condition action-value predictions. For example, when a stack of blocks falls over, they can re-build the stack of blocks as if they had just started the task. If they accidentally push granular media out of the target zone during sweeping, they then push it back in (Fig. 5). We hypothesize that equivariance to rotations and translations enable them to learn these recovery behaviors even with little data on multi-step tasks (see Tab. 1). We show examples of these emerging behaviors in our supplementary video.

Sample Complexity in Simplified Environments. Consider a simplified translation-only block-insertion task illustrated in Fig. 3, where no rotations are needed, and the block is initialized to the same location, with only the fixture location varying between environments. We investigate two variants of experts in this setting: (a) one that provides deterministic demonstrations where $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}$ are fixed relative to the block and fixture respectively, and (b) one that provides stochastic demonstrations where $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}$ are uniformly random sampled from the distribution

	deterministic				stochastic			
Method	1	10	100	1000	1	10	100	1000
Transporter Network	100	100	100	100	100	100	100	100
Form2Fit [24]	100	100	100	100	100	30.0	45.0	45.0
Conv. MLP	7.0	73.0	81.0	69.0	11.0	31.0	52.0	68.0
GT-State MLP	100	100	100	100	100	21.0	77.0	100
GT-State MLP 2-Step	100	100	100	100	53.0	70.0	100	93.0

Table 3. Simplified 2DoF (no rotation) block-insertion is harder to learn with demonstrations from a stochastic oracle than a deterministic one.
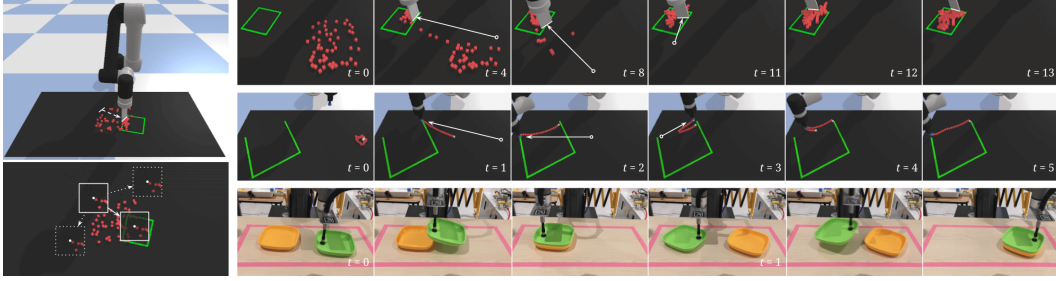
Figure 5. Transporter Networks can predict the desired spatial displacements of piles of small objects (left), which informs how to push them towards a target set. Our experiments show that it can learn visual feedback – enabling it to recover from mistakes (e.g., top row: pushing objects out of the target set at $t = 11$), or sequentially rearrange a deformable rope (middle row). Since it learns pick-conditioned placing, it can also stack plates with varying initial predicted pick locations from only 10 demonstrations on a real robot (bottom row), without needing any object-specific representations.

of successful actions. Intuitively, the stochastic demonstrator reflects a more realistic setting in which the expert is inconsistent. Tab. 3 displays sample complexity results on these tasks. Most baselines perform well with deterministic demonstrations, but begin to struggle with stochastic ones. Transporter Networks, by construction, work well in both cases. We show additional extrapolation experiments in the Appendix.

Learning 6DoF Pick-and-Place. We also test the SE(3) formulation discussed in Sec. 3.1 in the 6DoF environment shown in Fig. 4 with unseen target fixture configurations and stochastic demonstrations. On this more challenging task, our model demonstrates considerably better sample efficiency than the image-based baselines achieved on the strictly-easier 3DoF-only block-insertion task (Tab. 2 and 4). Additionally, while the hybrid discrete and con-

	6DoF block-insertion			
	1	10	100	1000
Transporter Net. SE(3)	38.0	76.0	84.0	91.0
GT-State MLP 3-Step SE(3)	0.0	1.0	1.0	5.0

Table 4. 6DoF block-insertion: our method generalizes with fewer demonstrations than GT-State MLPs.

tinuous SE(3) model is not able to 1-shot generalize as the discrete-only SE(2) model could, we still observe substantially better performance than GT-State MLP models (Tab. 4) which struggle in the larger observation and action spaces. For further analysis of this task and models, see the Appendix.

4.2 Real-World Experiments

We validate Transporter Networks on both the kit assembly and the sweeping tasks (shown in Fig. 1k and 1m) using real UR5e robots. For kit assembly, the robot uses a suction gripper and an industrial Photoneo PhoXi camera (for high resolution and accurate IR-depth). For sweeping, the robot uses a brush end effector and a low-cost consumer-grade Azure Kinect (for RGB-depth). Despite COVID-19

Task	Test %	# Samples
Mouthwash Kit Assembly	98.9	8141
Sweeping Piles of Go Pieces	98.3	6759

Table 5. Transporter Networks test performance on real robots with human demonstration data.

lockdowns preventing physical access, we were still able to perform real experiments using our Unity-based [45] UI that enables people to remotely teleoperate our robots to calibrate, collect training data, and run tests. For these experiments, teleoperators are tasked with i) repeatedly assembling and disassembling a kit of 5 bottled mouthwashes, or ii) sweeping piles of small Go pieces into a target goal zone marked with green tape, and autonomous resets using a scripted scrambling motion. Table 5 shows test performance of the models, and the amount of training data collected from 13 human operators in terms of transitions (i.e., pick and place actions for kit assembly, pushes for sweeping). We also show qualitative results of our robots in action for both tasks in our supplementary video. Further quantitative experiments were infeasible at time of submission due to COVID-19. Additional experiment details in the Appendix.

5 Conclusion

In this work, we presented the Transporter Network, a simple model architecture that infers spatial displacements, which can parameterize robot actions from visual input. It makes no assumptions of objectness, exploits spatial symmetries, and is orders of magnitude more sample efficient in learning vision-based manipulation tasks than end-to-end alternatives: from stacking a pyramid of blocks, to assembling kits with unseen objects, to pushing piles of small objects with closed-loop feedback. In terms of its current limitations: it is sensitive to camera-robot calibration, and it remains unclear how to integrate torque/force actions with spatial action spaces. Overall, we are excited about this direction and plan to extend it to real-time high-rate control, and as well as tasks involving tool use.

Acknowledgments

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research (JMLR)*, 2016.

[2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Conference on Robot Learning (CoRL)*, 2018.

[3] L. Manuelli, W. Gao, P. Florence, and R. Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *International Symposium on Robotics Research (ISRR)*, 2019.

[4] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih. Unsupervised learning of object keypoints for perception and control. *NeurIPS*, 2019.

[5] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. *Conference on Robot Learning (CoRL)*, 2020.

[6] X. Liu, R. Jonschkowski, A. Angelova, and K. Konolige. Keypose: Multi-view 3d labeling and keypoint estimation for transparent objects. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[7] O. Mees, N. Abdo, M. Mazuran, and W. Burgard. Metric learning for generalizing spatial relations to new objects. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[8] P. Jund, A. Eitel, N. Abdo, and W. Burgard. Optimization beyond the convolution: Generalizing spatial relations with end-to-end metric learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[9] P. R. Florence, L. Manuelli, and R. Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2018.

[10] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[11] P. Sundaresan, J. Grannen, B. Thananjeyan, A. Balakrishna, M. Laskey, K. Stone, J. E. Gonzalez, and K. Goldberg. Learning rope manipulation policies using dense object descriptors trained on synthetic depth data. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[12] H. Suh and R. Tedrake. The surprising effectiveness of linear models for visual foresight in object pile manipulation. *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2020.

[13] R. Kondor and S. Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *International Conference on Machine Learning (ICML)*, 2018.

[14] T. Cohen and M. Welling. Group equivariant convolutional networks. *International Conference on Machine Learning (ICML)*, 2016.

[15] R. Platt, C. Kohler, and M. Gualtieri. Deictic image mapping: An abstraction for learning pose invariant manipulation policies. *AAAI Conference on Artificial Intelligence*, 2019.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[17] Y. Yoon, G. N. DeSouza, and A. C. Kak. Real-time tracking and pose estimation for industrial objects using geometric features. *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[18] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis. Single image 3d object detection and pose estimation for grasping. *ICRA*, 2014.

[19] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[20] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox. Self-supervised 6d object pose estimation for robot manipulation. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[21] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[22] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics (T-RO)*, 2005.

[23] M. Gualtieri, A. ten Pas, and R. Platt. Pick and place without geometric object models. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[24] K. Zakka, A. Zeng, J. Lee, and S. Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[25] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager. "good robot!": Efficient reinforcement learning for multi-step visual tasks with sim to real transfer. *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[26] L. Berscheid, P. Meißner, and T. Kröger. Self-supervised learning for precise pick-and-place without object model. *IEEE Robotics and Automation Letters (RA-L)*, 2020.

[27] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel. Learning to manipulate deformable objects without demonstrations. *Robotics: Science and Systems (RSS)*, 2020.

[28] C. Devin, P. Rowghanian, C. Vigorito, W. Richards, and K. Rohanimanesh. Self-supervised goal-conditioned pick and place. *arXiv preprint arXiv:2008.11466*, 2020.

[29] M. Khansari, D. Kappler, J. Luo, J. Bingham, and M. Kalakrishnan. Action image representation: Learning scalable deep grasping policies with zero real world data. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[30] S. Song, A. Zeng, J. Lee, and T. Funkhouser. Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters (RA-L)*, 2020.

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

[32] D. Morrison, P. Corke, and J. Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *Robotics: Science and Systems (RSS)*, 2018.

[33] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[34] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[35] A. Zeng. *Learning Visual Affordances for Robotic Manipulation*. PhD thesis, Princeton University.

[36] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. *Conference on Computer Vision and Pattern Recognition*, 2019.

[37] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. *European Conference on Computer Vision (ECCV)*, 2016.

[38] R. Tao, E. Gavves, and A. W. Smeulders. Siamese instance search for tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[39] W. Luo, Y. Li, R. Urtasun, and R. Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[40] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[41] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *International Conference on Learning Representations*, 2016.

[42] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *International Conference on Machine Learning (ICML)*, 2010.

[43] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub Repository*, 2016.

[44] C. M. Bishop. Mixture density networks. *Aston University*, 1994.

[45] Unity Technologies. Unity game engine. Version: 2018.4.0f1. URL https://unity.com.

[46] T. Schmidt, R. Newcombe, and D. Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters (RA-L)*, 2016.

[47] R. Goroshin, M. F. Mathieu, and Y. LeCun. Learning to linearize under uncertainty. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[49] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. *ICRA*, 2018.

[50] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *European Conference on Computer Vision (ECCV)*, 2020.

[51] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[52] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

6 Appendix

The appendix consists of: descriptions of tasks and evaluation metrics in the Ravens simulation framework, additional experimental results, analysis, ablation studies, and real system details.

6.1 Simulation Experimentation, Real World Experimentation, and COVID-19

As introduced in the main paper, we execute experiments in simulation with Ravens as a means to provide a consistent environment for comparisons to baselines and ablations. Models evaluated in simulation are trained only on simulated data. Likewise models evaluated in the real-world are trained only on real data. In simulation, we avoid assumptions that cannot transfer to a real setup: observation data contains only 640x480 RGB-D images and camera parameters; actions are end effector poses (transposed into joint positions with inverse kinematics). The only exception to this is our GT-State baseline, which learns MLPs that take ground truth state information (e.g., object poses, bounding boxes, and color) as input – shown to be less sample efficient than Transporter Networks for many tasks. A limitation of Ravens is that the rendering software stack may not fully reflect the noise characteristics often found in real data: inaccuracies in camera calibration (e.g., intrinsics and extrinsics), noise in commodity depth sensors. Hence, we also test our method on a real system (details in Sec. 6.14) using teleoperated demonstration data, and present qualitative results in our supplementary video. Extensive quantitative experiments on real robots was difficult due to limited physical access during COVID-19.

6.2 Ravens Simulation Framework

In our simulation framework, Ravens, each task comes with a scripted oracle that provides expert demonstrations by randomly sampling the distribution of successful picking and placing actions – samples of which are made available to the learner. Ravens also has several attributes that make it useful for studying different areas that are beyond the scope of this work, including: 1) active learning, since algorithms can query an oracle during specific moments of uncertainty to improve learning efficiency, 2) reinforcement learning, since we provide reward functions that provide partial credit for all tasks (used only for evaluation in this work), and 3) active perception, since camera parameters (e.g., extrinsics) are defined by an action (static in our experiments, but could be dynamic), which provides an opportunity to study algorithms that improve learning through active control of the camera position and orientation.

During both training and testing, all objects (including target zones) are randomly positioned and oriented in the workspace. To succeed in the case where only a single demonstration is provided, the learner needs to be invariant to unseen configurations of objects. Information about the multi-modality of a task or its sequential permutations is only made available from the distribution of demonstrations. For example, moving disks in Towers of Hanoi may have only one correct sequence of states, but the distribution of how each disk can be picked or placed is learned from multiple demonstrations. Or, when palletizing homogeneous boxes, the arrangement of boxes across each layer on the pallet must be transposed, and boxes should be stacked stably on top other boxes already on the stack to avoid toppling. In our experiments, Transporter Networks are trained from scratch, without any pre-training from vision datasets. Exploring multi-task training for more efficient generalization to new tasks is interesting future work.
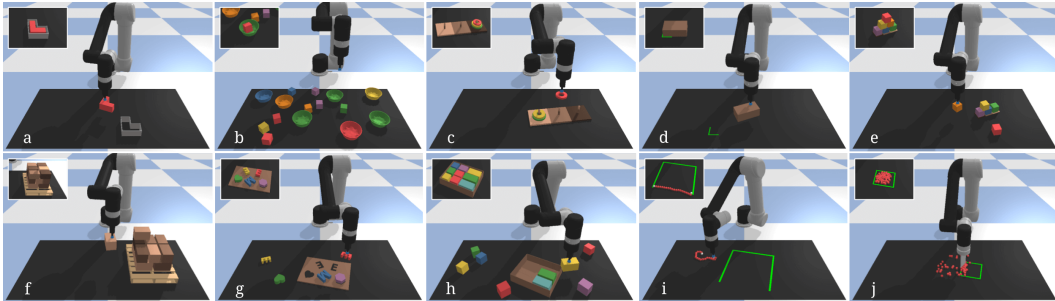

Figure 6. **Tasks:** (row-major order) block-insertion, place-red-in-green, towers-of-hanoi, align-box-corner, stack-block-pyramid, palletizing-boxes, assembling-kits, packing-boxes, manipulating-rope, sweeping-piles. Goal states (not provided to learners) are shown in top left corner of each image.

6.3 Ravens-10 Benchmark Task Details

A set of 10 tasks (illustrated in Fig. 6) comprise our Ravens-10 benchmark. In each of these tasks, the agent acts with motion primitives parameterized by a sequence of two SE(2) end effector poses. Note that although the two motion-primitive-defining poses are in SE(2), manipulation behaviors that are in 3D can still be achieved (such as stacking blocks on top of each other) due to motion primitives that involve out-of-plane movement.

block-insertion: picking up an L-shaped red block and placing it into an L-shaped fixture with a corresponding hole. The block is 8cm in diameter while the hole is 9cm in diameter. The block and fixture configurations are randomly sampled in the workspace – rejected if in collision with each other, or if partially out of bounds in the workspace.

place-red-in-green: picking up red blocks and placing them into green bowls amidst other visually distracting objects (e.g., randomly positioned blocks and bowls of different colors). There can be multiple red blocks and/or multiple green bowls, with a random chance for more bowls than blocks. This task requires handling multiple candidate picking and placing locations. Since bowls are twice as large as the blocks, this task does not require precision. Red blocks and green bowls are first added to the workspace, then 10 objects (blocks or bowls of other colors) are randomly spawned in collision-free configurations.

towers-of-hanoi: sequentially picking up disks and placing them into pegs such that all 3 disks initialized on one peg are moved to another, and that only smaller disks can be on top of larger ones. Completing this task requires sequencing 7 pick-and-places of the disks in a specific order. During evaluation, no information about which disk should be picked next is provided – the approach needs to infer this from contextual visual cues e.g., the positions or absence (due to occlusions) of other disks. This tests the capacity of the learner to infer the sequential semantics of Markovian vision-based manipulation tasks. Fig. 7 illustrates this task, its ground truth sequence, picking action distribution, and placing action distributions (unimodal for translation-only placing, or spiral shaped for the SE(2) placing). The base of the 3 pegs are randomly configured in the workspace, and disks are initialized on the peg on the lightest side of the base.
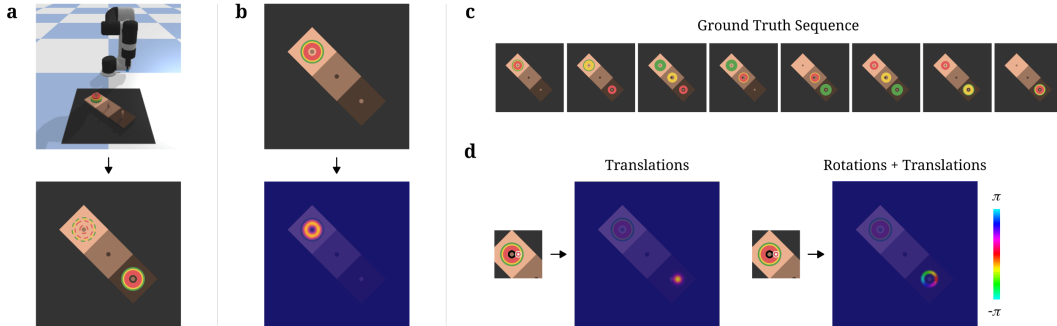

Figure 7. In Towers of Hanoi (a), the task is to sequentially pick-and-place 3 disks from the first peg to the third peg without placing a larger disk over a smaller disk (c). In each step, the distribution of successful picks is ring-shaped (b), and the distribution of successful pick-conditioned placements (d) is unimodal with translation-only placing, or spiral-shaped with SE(2) placing (projected onto a ring for visualization, where the ring traverses different rotations). The nature of this task evaluates the capacity of Transporter Networks to recover non-trivial spatial action distributions, and to sequence actions by leveraging contextual visual cues.

align-box-corner: picking up a randomly sized box and repositioning it on the tabletop to align one of its corners to a green L-shaped marker labeled on the tabletop. This task requires precision, and the ability to generalize to new box sizes. Aligning any of the box's corners to the green marker yields task success. The marker and box are randomly configured in the workspace such that the box does not overlap the marker.

stack-block-pyramid: sequentially picking up 6 blocks (each with a different color) and stacking them into a pyramid of 3-2-1. The bottom 3 blocks can only be purple, green, or blue blocks. The middle 2 blocks can only be orange or yellow blocks. The top block can only be red. A successful pyramid is invariant to the permutation of blocks within each row, but not between rows. The blocks and base of the pyramid are randomly spawned in collision-free configurations.

palletizing-boxes: picking up fixed-sized boxes and stacking them on top of a pallet. To mimic real stability constraints, the arrangement of boxes should be transposed across each layer of the pallet. There are a fixed number of boxes – each one is spawned into the workspace prior to each pick-and-place action.

assembling-kits: picking and placing different shaped objects onto corresponding locations on a board (visually marked with corresponding silhouettes of the objects). For each episode, 5 objects are randomly selected with replacement from a set of 20: 14 used during training (left), and 6 held out for testing (right). This task requires precise placing and generalizing to new objects by learning the concept of "how things fit together" [24]. The kit and 5 objects are randomly spawned in collision-free configurations.
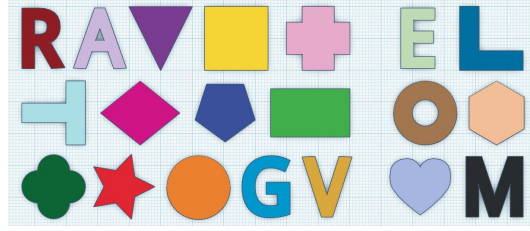

Figure 8. Kitting objects used in training (left 14) and testing (right 6).

packing-boxes: picking and placing randomly sized boxes tightly into a randomly sized container. This task is the hardest one in the benchmark, which requires not only relying on visual cues to tightly fit objects together, but also implicitly learning to order the objects in ways that maximize container coverage. During initialization, the container size is first randomly sampled, then randomly partitioned using a binary search tree (with minimum size of 5cm along each dimension). Each partition then parameterizes the size of each box, individually spawned into the workspace with a collision-free configuration.

manipulating-rope: manipulating a deformable rope such that it connects the two endpoints of an incomplete 3-sided square (colored in green). The cable is initialized as a randomly formed pile in the workspace, which needs to be untangled then aligned with the square. This task requires sequencing actions with closed-loop visual feedback to incrementally adjust the rope. The rope is implemented as a set of 20 articulated spherical beads, randomly dropped onto the workspace. The 3-sided square is randomly configured in the workspace.

sweeping-piles: pushing piles of small objects (randomly initialized) into a desired target goal zone on the tabletop marked with green boundaries. This sequential task (inspired by [12]) requires sequencing pushes with closed-loop visual feedback and handling piles of small objects. In this task, the suction end effector is swapped with a spatula. The small objects and the target goal zone are randomly initialized in the workspace.

6.4 Additional Tasks

In addition to the Ravens-10 tasks, we used two other tasks in the paper for analysis. The first was to provide a simplified test environment for testing the difference between stochastic and deterministic demonstrations (Sec. 4), and interpolation/extrapolation experiments with a small enough amount of degrees of freedom such that they could be plotted (Sec. 6.10). The second task was to test the SE(3) variant of Transporter Networks.

2DoF simplified block-insertion: This task is identical to the *block-insertion* task, except: (i) the block is always in the same staring configuration, and (ii) the fixture varies only in translation. The environment initialization has 2 degrees of freedom total.

6DoF block insertion: This task is identical to the *block-insertion* task, except that the fixture varies in 6 degrees of freedom, requiring SE(3) placing. In addition to the table-plane degrees of freedom, the fixture also varies in height (z) as well as the two rotational degrees of freedom out of the plane, which may be represented by *roll* and *pitch*. The block varies only in SE(2). The environment initialization has 9 degrees of freedom total.

6.5 Evaluation Metrics

Performance on each task in simulation is evaluated in one of two ways:

Pose: object translation and rotation to target pose is less than a threshold $\tau = 1$cm and $\omega = 15°$ respectively. If completing a task requires sequencing multiple actions, then each successful action (where an object is moved to its correct pose) returns a partial reward $r = \frac{1}{\text{\# of actions}}$. Total rewards for completing a task always sums to 1. Object symmetries are accounted for. Tasks: block-insertion, towers-of-hanoi, place-red-in-green, align-box-corner, stack-block-pyramid, assembling-kits.

Zone: ratio of object(s) in the target zone. We discretize each object's 3D bounding box into 2cm^3 voxels. Total reward is the fraction of total voxels in the target zone $r = \frac{\text{\# of voxels in target zone}}{\text{total \# of voxels}}$. Tasks: palletizing-boxes,

packing-boxes, manipulating-cables, sweeping-piles. Achieving total reward $r=1$ on palletizing-boxes and packing-boxes requires tightly fitting all objects into the target zone (i.e., it is exponentially more difficult to achieve $r=0.9$ than it is to achieve the first $r=0.5$).

6.6 Data Augmentation

As is common with deep learning methods, we find that data augmentation can substantially benefit generalization performance for Transporter Nets. To control the experimental comparisons with baselines in this paper, all methods used an identical data augmentation distribution. Specifically, a random SE(2) transform, $T_{\mathcal{W}}^{aug}$, relative to the world frame $\mathcal{W}$ was sampled, and all observations (whether the images, or the object poses for the GT-State methods) were adjusted as if the data was observed from the augmentation frame aug rather than the world frame. This augmentation makes the assumption that the absolute position of objects on the table plane does not matter for the tasks – this was indeed the case for all tasks shown. Note that this distribution provides augmentation along the 3 degrees of freedom of SE(2), but we do not apply other augmentations such as imagining different relative configurations of objects – this would not be a feasible assumption for certain tasks that involve multiple objects.

6.7 Additional Baseline Details

Form2Fit [24] is a recent method for pick-and-place (also similar to Devin et al. [28]), using a matching network that computes dense visual descriptors [9, 46] to associate picking actions to placing actions. We use an author-provided implementation of Form2Fit with 43-layer ResNet backbones of similar parameter count for a fair comparison.

ConvMLP is a model architecture first proposed in [1] for learning control policies from pixel input – commonly used in end-to-end literature. While typically this architecture is used for real-time continuous control and was originally used with a joint torque action space, we tried it here for our motion-primitive-based setting with a spatial action space. The input images are the same as that of Transporter Networks and Form2Fit: top-down reconstructed views from RGB-D data. The architecture consists of a series of three convolutional layers, followed by spatial soft(arg)max [47] and a multi-layer perceptron to regress a 6-dimensional vector of the two SE(2) end-effector poses. The architecture used was almost exactly as in [1], but with the following changes: (i) we used channel depths of 64, 64, 64, rather than the 64, 32, 16 used in the original paper, (ii) we used MLP layer widths of 128, 128, (iii) we used the MLP to regress mixture densities [44] rather than direct regression in order to handle multi-modality, and (iv) we added depth image input to the network, by replicating depth along 3 channels, processing it with one separate 7x7 convolutional layer, and then fusing with RGB after the first layer via concatenation. The addition of depth image input into the model was to be more directly comparable with the other image-based options, which also used depth. As in [1], we use ImageNet-pretrained weights from GoogLeNet [48] for the first RGB convolutional layer, which we find to aid in stable training and generalization. While this network is 10x shallower (3 layers) than the network used for Form2Fit and Transporter Nets (43 layers), we tried to train the same 43-layer backbone network used in Transporter Networks with a spatial soft(arg)max and MLP, but found that the shallower 3-layer network achieved better generalization at least for our tested hyperparameters. Since many tasks involved multi-modal policies, it was necessary to train the network in a way that handles this multi-modality. Mixture Density Networks (MDN) [44] have previously been used for end-to-end manipulation control, for example in [49]. As in [44], we trained these with a negative log likelihood loss. We used a mixture of 26 Gaussians, with a training temperature of 2.5. To match translation and rotational units on a common scale, we equalized $1\ m$ translation with 0.1 radians. We also found improved generalization by taking the features $\mathbf{z}$ extracted by the spatial soft(arg)max, and concatenating $\mathbf{z} \leftarrow [\mathbf{z}, \sin(\mathbf{z}), \cos(\mathbf{z})]$ as input to the MLP – this was inspired by [50, 51]. We experimented with dropout in the MLP to add regularization, but found this to decrease generalization; the noise from the convolutional feature extraction was enough to regularize the MLP layers. The models were trained with the Adam optimizer for 20,000 steps, batch size 4, with a learning rate of $2e^{-4}$.

GT-State MLP is a model architecture for learning control policies directly from ground truth state (i.e., object poses and 3D bounding boxes) as input, similar to the baseline used in [10]. In the real world, this would be analogous to assuming perfect object pose estimation was provided by an external vision system. The model shares an identical MLP to the ConvMLP model described above, except the input $\mathbf{z}$ to the model was all degrees of freedom of the environment (object poses, bounding boxes, and in the case of the place-red-in-green task we also added the RGB color of each object as input). The bounding boxes are used for the 3 tasks (palletizing-boxes, align-box-corner, and packing-boxes) that involve boxes that

vary over their length, width, and height, so the bounding boxes exactly parameterize these shapes. Some environments have a variable number of objects, and to handle this we padded the input vector with zeros to match the largest cardinality seen in the environment sampling. We used dropout in the MLP layers with a dropout rate of $p=0.1$, which we found to improve generalization. The models were trained with the Adam optimizer for 40,000 steps, batch size 128, with a learning rate of $2e^{-4}$.

GT-State MLP 2-step is composed of two MLP networks, each of which are almost identical to GT-State MLP. The first regresses only the first SE(2) pose, and the result of this is added to the observation vector of the second network, which regresses the second SE(2) pose. We used this model to experiment with whether the benefit of Transporter Networks was due to their sequential model cascade, where the second model is conditioned on the first (i.e., pick-conditioned placing). We experimented with training the second MLP model on noisy regressions from the first model – but perhaps because our models were sufficiently regularized with $p=0.1$ dropout, we found this to not provide measurable advantages over simpler teacher forcing, in which the second model was only trained on the conditioning from the training data.

GT-State MLP 3-step SE(3) is almost identical to the 2-step model, except a third model is conditioned on the outputs of both the first and second model, and the additional out-of-plane degrees of freedom (height, roll, pitch) are regressed. We experimented with also using just one GT-State MLP for the SE(3) action space task, but found similar performance. Additionally, this model took in as input not only the SE(2) degrees of freedoms of objects, but their full SE(3) poses, represented with roll, pitch, yaw for the rotational degrees of freedom. We experimented with other forms of rotational observation, such as observing three 3D points on the object, which does not have 2π wrapping issues, but for the 6DoF task tested we found similar performance to the roll, pitch, yaw observation.

6.8 Additional Analysis of Sample Efficiency and Baseline Comparisons

Here we add additional analysis of the results in Tab. 2 and Tab. 3, which show the sample efficiency of baselines trained from stochastic demonstrations for each task, and evaluated on unseen test settings.

The results suggest that Transporter Networks accelerate end-to-end learning on the wide variety of tasks tested. We believe this is due to the inductive bias of the model, which i) incorporates a high-level prior (that manipulation involves rearranging 3D space), and ii) uses the spatial structure of the visual input to exploit symmetries in the data to better learn useful correlations.

Among the two image-based baselines, Form2Fit performed better than ConvMLP on most tasks – all except the sweeping-piles task, which may be especially hard for the descriptor-based Form2Fit model to work well on the pile of small items, since no item in the pile is significantly different than the others, but rather the policy should be a function of their collective distribution. While for some tasks, like the place-red-in-green task, the Form2Fit model does comparably well to Transporter Networks, it struggles on tasks that require either more precision, or multi-step tasks. The Tab. 3 results for Form2Fit are also especially interesting – although the model can 1-shot generalize with 100% success on the simplified block insertion task, it starts to struggle fitting additional stochastic demonstrations. The ConvMLP model for the most part sees a monotonic increase in performance with additional demonstrations, but often does not generalize as precisely as other models in this small-data regime. In all scenarios tested, Transporter Networks outperformed both image-based baselines.

For the state-based baselines, these experiments also show that Transporter Networks often require less data than MLP policies trained from ground truth state (e.g., list of object poses, bounding boxes, and color). Of course, it is possible to have a policy that consumes the ground truth state of the scene and performs perfectly on all tasks – indeed, the scripted expert policies do exactly this. For learning generalizable policies from demonstration, however, our experiments suggest that directly reasoning over the spatial configuration of visual cues may instead provide grounds for discovering more generalizable patterns for manipulation. The simplest example of this may be that it's easier to learn, from few demonstrations, a policy that is "pick a red pixel" and "place in a green pixel", rather than to regress these values in a continuous space, from a vector of continuous values. Other additional cues that may be easier to learn directly from images include: aligning edges and corners when packing boxes and assembling kits, or matching gripper shapes to object surfaces for grasping.

In addition to the difference of using either images or ground-truth state, it is also important to keep in mind that the GT-State MLP model fundamentally differs in a couple other ways as well, including that its action space is continuous rather than discrete, and it is trained with a multi-modal regression loss rather than a classification loss. It would be interesting to consider other model architectures that consume

ground-truth state information and better model intra-object relationships and variable-dimensionality input, such as graph neural networks (GNNs), and also consider models that naturally include some translation equivariance, which MLPs do not. Given the ubiquitity of MLPs in continuous regression tasks, however, and its proximity to the ConvMLP baseline, we consider our chosen MLP model to be a highly relevant baseline.

One additional note is that for the sweeping-piles task, we observe that a Transporter Network which is constrained to choose a sequence of two translation-only poses, where the rotation is determined as being orthogonal to the line between these two poses, works better than allowing the model to choose a rotation at each pose. This may be due to the specifics of the sweeping motion-primitive used for this task, but it also may indicate that the rigid-transformation bias in the model has its limits. The pile of objects does not get perfectly rigidly transformed, but rather has more complex dynamics. We plan to investigate this further in future work.

6.9   SE(3) Transporter Networks: Architecture Discussion and 6DoF Task Analysis

Network Architecture Considerations. We experimented with multiple versions of extending Transporter Networks into 6-degree-of-freedom, SE(3) action spaces. The hybrid discrete/continuous model we have presented in the main paper provides a number of nice features, and demonstrates the flexibility of the transport operation. It would also be interesting to try pure-discrete classification models, but fully sampling over the entire action space becomes burdensome with higher degrees of freedom. In the first SE(2) step, convolution can be used to efficiently imagine all overlays for the image-width and image-height dimensions, requiring sampling only over the in-plane rotations. However, the remaining 3 degrees of freedom in SE(3) are all orthogonal to the in-plane image dimensions. While a discrete version could be done with rotating 3D voxel grids, this might present memory and sparsity challenges. Instead, our presented choice of using continuous regression to directly regress the remaining degrees of freedom is an attractive one – essentially building on the same architecture as the SE(2) version with little modification.

Here we provide additional details of our specific evaluated SE(3) approach with results shown in Tab. 4. As discussed in Section 3.1, all SE(2) degrees of freedom are resolved identically to the SE(2)-only models. An additional model then handles the remaining SE(3) degrees of freedom: roll, pitch, and z. The additional model is almost identical to the SE(2) pick-conditioned placing model, but is trained to do three independent regressions rather than one classification. In this way, it still benefits from the imagined overlay of the transport operation as an attention mechanism. To adjust the architecture for this purpose, we first increased the output dimensionality to give $d\!=\!24$ channels, rather than the $d\!=\!3$ channels used for cross-correlation in the the classification models. Additionally, rather than cross-correlating all 24 channels at once and reducing to an $\mathbb{R}^{H \times W \times k}$ tensor, for image height H, width W, and in-plane rotation discretizations k, we cross-correlate three distinct subsets of 8 channels to give an $\mathbb{R}^{H \times W \times k \times 3}$ tensor, which retains sufficient degrees of freedom to regress 3 independent values. These 3 values in this tensor can be directly trained to do regression, but we find precision improvements if we add a small amount of non-linearity. The small non-linearity mapping we use, $f(\cdot)$ in Sec. 3.1, is three independent MLP networks, which each take in one value and regress one value, and are each 3-layer, 32-width MLP networks with ReLu activations. This is similar to 1x1 convolutions on the spatial feature maps, but with non-linearity. We also find increased generalization if, rather than applying the regression loss only to the correct 1-hot label in $\mathbb{R}^{H \times W \times k}$, we also apply regression loss to a small window in that region. We used a window of 7x7 pixels, and +/- 1 discretization in k.

We tried variations to the above described approach, many of which worked reasonably well. One option, for example, is to use only one pick-conditioned placing network, which produces an $\mathbb{R}^{H \times W \times k \times 4}$ tensor, of which for the last tensor dimension, 1 channel is used for classification loss and the others are used for regression. We found, however, both the classification loss and the regression loss to benefit from using separate networks–this is potentially due to differences in the training dynamics of the classification and regression losses. We also tried various different methods of extracting the 3 regression channels from the spatial feature maps, including using spatial soft(arg)max from the overlaid crops followed by MLPs, and using concatenation rather than cross-correlation before the MLPs. Each of these worked reasonably well.

Additional 6DoF Task Performance Analysis. Here we expand on analysis of the results presented in Tab. 4. One important observation is that it was not clear, prior to these experiments, that the model would be able to successfully generalize due to its multi-step approach. In particular, the SE(2) placing step occurs before control of the remaining degrees of freedom. While this might be more challenging for different objects such as long and thin ones, we find that for our tested environments, we are able to get above-90%

generalization on this challenging task given 1,000 training examples. Introspection into the accuracy of the different degrees of freedom of this model provides interesting comparisons as well. Specifically, for $n=1,000$ training examples, the mean absolute errors (MAE) for roll/pitch regressions were each in the range of 3 to 4 degrees, whereas the MAE for in-image-plane rotation (yaw) were in the range of 10 degrees, and most often contributed to unsuccessful deployments in test environments. This suggests that future work building on this model could investigate an iterative approach in which the initial degrees of freedom would once again be optimized, potentially iteratively N times, and potentially with increasingly fine resolution discretization. Additionally, it should be noted that in our tested model the discrete step occurs before the regression step, and so in a sense has a more challenging task, since there may not be good alignment of the remaining degrees of freedom. Additionally, we only used SE(2) augmentation as discussed in Sec. 6.6, since the presence of gravity does not make all roll/pitch-augmented worlds equal, and so the remaining SE(3) degrees of freedom do not benefit from the same level of augmentation. We do find however that even in the $n=1$ demonstrations case, with 38 % success, the SE(3) model behaves as if it uses one azimuth angle, consistent with a single roll/pitch being rotated around the z axis, but this is actually a manifold of different roll/pitch values. This shows the benefit of SE(2) augmentation even for the remaining SE(3) degrees of freedom.

Also as shown in Tab. 4, our tested GT-State MLP model significantly struggled with the 6DoF SE(3) task, much more so than it did with the 3DoF SE(2) task. Investigation into the cause of the challenge shows that this is primarily due to the additional degrees of freedom in the observation and action spaces. In particular we can see this in the tested 3-step model (described in Sec. 6.8), which often struggles to precisely regress the x and y translational degrees of freedom even with $n=1,000$ demonstrations, which was not a problem in the simpler task. Additionally, if we take this task's data, but reduce the observation and action spaces to only be SE(2), then with $n=1,000$ demonstrations the model is once again able to precisely regress the x and y translations. We hypothesize that the mix of the stochastic demonstrations and the full 6-DoF placing is challenging for the model to learn a generalizable mapping with this amount of data.

6.10   Interpolation and Extrapolation

To gain a better qualitative understanding of the generalization abilities of Transporter Networks and the tested baseline models, we provide Fig. 9. Similar to the analysis in [10], this shows the spatial distribution of training configurations and successful/unsuccessful testing configurations–in this case, for translation-only block insertion with stochastic demonstrations. For each method we visualize its ability to interpolate and extrapolate from its training samples to handle new unseen poses of the target fixture.

As can be seen from these generalization plots, on this task Transporter Networks is capable of extrapolating to new testing episodes with just one demonstration. The dense deep feature transport operation enables the optimal placement of the block in the fixture to look similar under different picking and placing actions (exploiting translational equivariance of the task). This generalizes to new object configurations better than Conv MLP, which seems to only successfully interpolate around the majority of training episodes (often collapsing towards one mode of the training data).

Form2Fit is also capable of extrapolating from a single example, since both its picking and placing networks (also modeled by FCNs) are translationally equivariant – such that the block insertion task amounts to detecting the block and fixture under different translations. Once the number of demonstrations increase, however, the matching network is forced to learn pixel-wise descriptors that index multiple possible picking actions to multiple possible placing actions. In the setting with stochastic demonstrations, the positive-pair samples available for training the matching network are sparse and limited only to the pick-and-place action pairs from demonstrations – making it difficult to geometrically regularize the descriptors uniformly over the pixels on the surface of the block and the fixture. Minor mis-matches during testing can also lead to a slight misalignment of the block, which leads to poor placement and task failure. This suggests that Form2Fit's matching network requires substantial training data to successful converge to a meaningful solution. The distribution of successful episodes with Form2Fit are also uniformly distributed over the workspace, which further confirms that its failures come largely from mismatching pick-and-place actions, rather than its inability to spatially interpolate or extrapolate.

In the low data regime (with 10 demonstrations), GT-State MLP seems to overtrain to correlations that do not necessarily enable task success, e.g., that picking locations are conditioned on the location of the target fixture (see distribution of failed pick locations and fixture locations). By dividing GT-State MLP into two steps – one for picking and one for placing, we see less issues with incorrect correlations. However, the inability to extrapolate too far beyond the distribution of training data still remains (e.g., see failure

Figure 9. Depictions of the generalization ability of different models on the simplified translation-only block-insertion task. Each episode is visualized as a mark on the workspace (representing the pose of the fixture) – color-coded by task success (blue for success, red for failure). Environment configurations in the training set are shown in gray, and the convex hull of the training samples are drawn with a dotted line. Testing episodes inside the convex hell represent interpolation; those outside represent extrapolation. Plot inspired by [10].

testing configurations of fixture for GT-State MLP 2-Step with 100 demonstrations, which also caused an incorrect pick point prediction).

6.11 Ablation Studies

Tab. 6 also reports the performance of various ablations of Transporter Networks:

No-Transport uses two FCNs: one to predict a picking heatmap, another to predict placing heatmaps (one per discretized rotation). The transport model is not used here, hence there is no pick-conditioned placing. For simpler tasks, such as place-red-in-green, this could suffice if the picking model predicts higher values on red pixels, and the placing model predicts higher values on green pixels. Or, for block-insertion, if the picking model detects the same location on the block, and the placing model detects the same corresponding location and orientation in the fixture. However, it is much more challenging to execute tasks that require additional visual reasoning, such as aligning unseen box corners. For palletizing and packing boxes, the learned policies are generally capable of placing a few of the boxes into the target zones, but fail to align them to each other tightly to achieve higher completion scores above 55%.

Per-Pixel-CE-Loss is a variant of the Transporter Network, where the cross-entropy loss over the entire image (representing a single probability distribution) is replaced with per-pixel cross-entropy, where each pixel represents its own probability distribution. During training, for every demonstration pick-and-place: the picking and placing pixels unraveled from the data are chosen as positive samples for the models
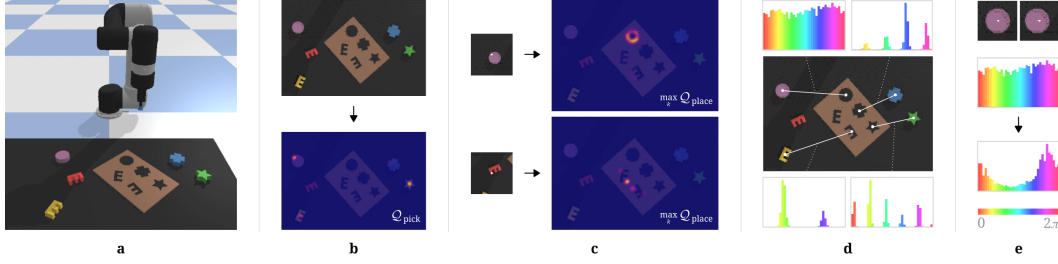
Figure 10. **Visualizing Transporter Network Predictions.** For the kit assembly task (a), Transporter Network picking predictions show 3 dominant modes on 3 different objects (b) which are not necessarily centered on the object (which may reflect biases learned from the distribution of training data). If a pick is off-center e.g., on the circle block, (c, top), the Transporter Network correctly makes placing predictions along the edges of the target placement location – which is visualized as a ring on $\mathcal{Q}_{place}$ maxed over rotations. If a pick is on the corner of the unseen E-shaped object (c, bottom), then two dominant modes lie on the correct corresponding locations of their target placements – but also with false positive modes from geometric symmetries in the letter E. We also show modes of predicted placing values over rotations for pick-and-place locations centered on the object and fixture, which appear uniform for circle, 4 modes for the cross shape, 5 modes for star shape, 2 modes for the unseen E-shaped block (dominant one being correct). We also visualize how the predicted placing values over rotations change from uniform, to unimodal, when shifting one pixel to the right from the center of the circle.

respectively, while a fixed number of 100 random negative pixels are sampled from everywhere else. In general, our experiments suggest that the model i) takes more training iterations to converge, and ii) requires more hyper-parameter tuning to balance the ratio of positives to negatives to regularize the loss over time.

Method	block-insertion				place-red-in-green				towers-of-hanoi				align-box-corner				stack-block-pyramid			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Per-Pixel-CE-Loss	89.0	88.0	91.0	90.0	87.5	100	100	100	29.6	57.0	57.9	64.6	46.0	65.0	73.0	77.0	23.0	28.0	33.7	36.0
No-Transport	5.0	8.0	6.0	8.0	81.5	95.7	100	100	2.1	3.6	4.7	3.3	9.0	7.0	11.0	15.0	8.0	13.2	16.0	25.8

	palletizing-boxes				assembling-kits				packing-boxes				manipulating-rope				sweeping-piles			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Per-Pixel-CE-Loss	40.3	41.5	39.1	42.7	12.8	48.0	50.0	52.6	44.8	58.1	58.9	68.1	4.9	29.7	28.8	38.9	30.3	29.9	37.5	39.6
No-Transport	44.3	50.4	53.6	54.1	0.6	4.0	1.8	2.8	11.3	40.1	42.3	53.5	33.5	35.3	38.8	40.0	13.7	18.8	28.8	38.1

Table 6. **Ablative comparisons.** Task success rate (mean %) vs. # of demonstration episodes (1, 10, 100, or 1000) used in training.

6.12   Training Convergence

We train Transporter Networks with Adam [52], using fixed learning rates of 10^{-5}. Our models are trained from scratch (i.e., random initialization) in Tensorflow. On the right, we show an example convergence plot over training from 100 demonstrations of manipulating rope. Compared with image-based baselines, Transporter Networks generally converge faster – within a few thousand training iterations, i.e., one or two hours of wall-clock time.
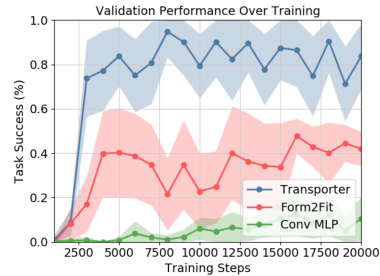
6.13   Visualizing Transporter Network Predictions

Fig. 10 visualizes Transporter Network dense pixel-wise predictions of $\mathcal{Q}_{pick}$ and $\mathcal{Q}_{place}$ for the kit assembly task (trained with 1,000 demonstrations), and how the predicted modes change with respect to the locations of objects, their shapes, and conditioned picks. These visualizations show that due to rotation and translation equivariance, our method can quickly learn modes that reflect symmetries in the data.

These visualizations also show an interesting failure mode, in which predicted placements of the unseen E-shaped object returns several 180 degree flipped false positives (see Fig. 10c). This is because the learned features from this task rely strongly on local geometry. Since the E shape is a new unseen object, a flipped E looks similar enough to return a false positive signal there (though not enough to be the top mode).

6.14   Real-World Experiments Details

To demonstrate the real-world performance of Transporter Networks, we apply our approach to two challenging real-world tasks: assembling kits of mouthwash bottles with pick and place, and sweeping piles of small Go pieces with pushing. These tasks are shown in Fig. 11.

For kit assembly, we sequentially pick 5 small mouthwash bottles from a "bin" of identically shaped items and place each into a "kit" with 5 corresponding indentations for placement. This is a challenging task due to the multi-modality and ambiguity of the pick location (many identical items in the bin to pick from), as well as the tight millimeter tolerances of the destination kit. It is designed to mimic a production robotic task of placing items into a blister pack for consumer packaging. Kit assembly and disassembly are treated as separate tasks – both use the same Transporter Network architecture with autonomous switching between the two tasks at inference time. As shown in Table 5, our model achieves 98.9% on kit assembly, trained with 8,141 pick and place actions recorded from human demonstrations. Picking success and disassembly success are both above 99%.

For sweeping, we sequentially push piles of small Go pieces into a target goal zone, marked with green tape. Upon completion of the task, the robot autonomously resets with a scripted scrambling motion. This task is challenging as it tests the capacity of the model to sequence actions using closed-loop visual feedback, and to do it efficiently. As shown in Table 5, our model achieves 98.3% on this task, trained with
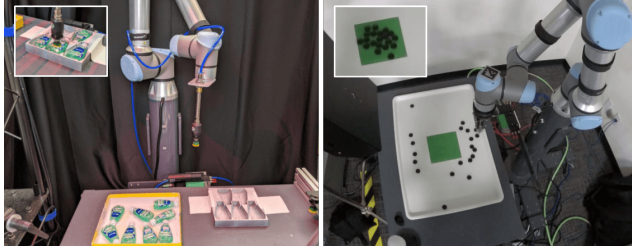

Figure 11. **Real-world tasks.** *Left:* assembling kits of mouthwash bottles, *right:* pushing piles of small Go pieces.

6,759 pushing actions recorded from human demonstrations. The average number of sweeps needed to complete the task is 20 ± 2.5 sweeps, with the fastest run being 16 sweeps and the slowest 25. For reference, the average for human teleoperation is 18.1 sweeps.

To train our system for both tasks, we use human demonstration data captured by a remote teleoperation interface. The sub-optimal, biased and noisy distribution of human demonstrations presents a further challenge for robotic policy learning. In addition, 13 teleoperators of varying skill and performance level provide training data, which further increases the variance of ground-truth actions that we train from and subsequently increases the difficulty of the learning task.

Hardware Setup

The hardware setup for kit assembly is shown in Fig 12. Our experiments make use of 2 Universal Robots UR5 workstations (one for kit assembly and one for sweeping), each with an industrial Linux PC, vacuum system, Piab suction cup or brush end-effector, and depth camera (statically mounted above the workstation). To test the robustness of our method to diverse data sources, we use high quality depth camera observations from a Photoneo PhoXi Model M for kit assembly, and those from low cost consumer-grade Azure Kinect for sweeping. The Photoneo camera supplies depth (with 0.1mm rated depth precision) and a greyscale IR image, both at 1032×772 resolution. The Kinect camera supplies depth as well as a color RGB image at 1280×720 resolution.

Figure 12. **Hardware setup.** a) UR5e Robot, b) Piab suction cup, c) Photoneo camera, d) Linux PC, and e) air compressor and vacuum generator.

To calibrate the camera within the robot coordinate frame we use a two step procedure. Firstly, the intrinsics of the camera are calibrated by capturing multiple images of a large flat QR-Code panel in varying orientations, and we use OpenCV to calculate the camera intrinsics. Secondly, to calibrate extrinsics we attach QR code tags to the UR5 wrist joint and we capture multiple images of the robot in random end-effector poses. These images are then used to solve for the position of the robot base as well as the offset of the QR Tag to its respective joint with a stochastic optimization procedure.

A Linux system embedded with the robot gathers robot and camera data and forwards it to the teleoperator over the internet. Any commands passed to the robots are queued up to be processed one-by one. All data is logged in real-time via network to a cloud storage database for model training and evaluation.
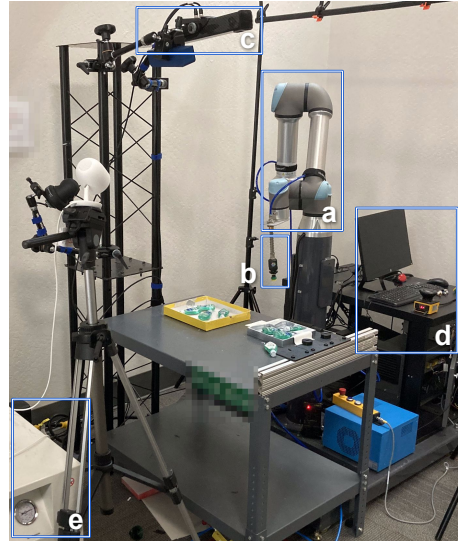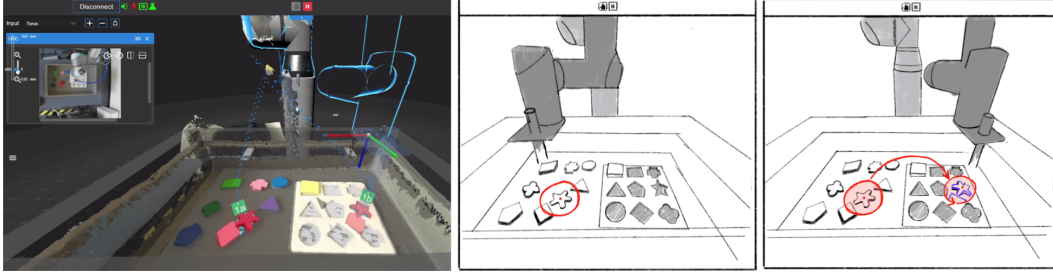
Figure 13. **Data Collection UI.** *Left:* Unity-based UI. *Middle:* The operator selects the area where the object is to be picked up. *Right:* The operator can adjust the placement of the object and rotate it. The depth image around the pick-up point is overlaid and helps the operator align the placement.
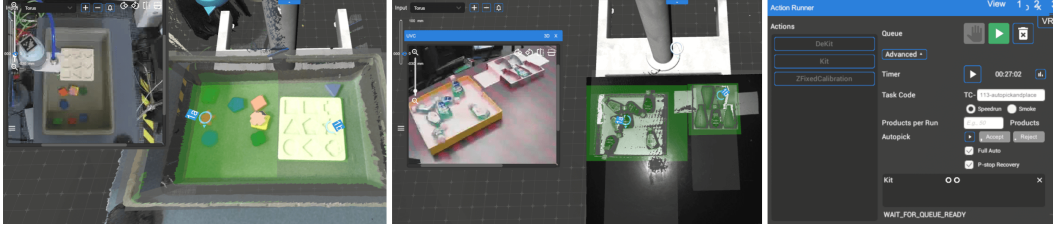

Figure 14. **Autonomous Mode UI.** User inputs the task to execute (by unique task code), and presses the play button to begin autonomous execution.

Data Collection

We designed a 2D User-Interface (UI) scripted in Unity [45] in order to control the robot workstations remotely. An overview of this UI is shown in Fig 13. Features of this UI include: configuration of the robot workcell, movement and actuation of the suction end-effector, configuration of parameterized action trajectories, calibration of the robot and control of the robot using the same action parameterization used to train our model.

A standard teleoperation session is as follows. The operator views a 3D rendering of the workstation depth data. The synthetic camera view can be rotated and translated using mouse movements as desired by the operator. The operator determines with mouse clicks which object to pick by selecting the intended pick-up point on the rendered depth geometry. A region of the point cloud around the pick location is then "attached" to the mouse cursor, where it is rendered transparent and overlaid on the existing depth (illustrated in Fig 13). The operator is able to translate this geometry with movements of the mouse and the single rotational degree of freedom is controlled by the mouse wheel. Once the operator moves the object into the desired location, an additional mouse click will select the 3DOF place location. This interface allows the operator to visualize the intended 3D configuration of the pick-and-place operation before the action is executed. The discrete teleoperator pick-and-place locations are then translated into dense robot trajectories using a Cartesian controller, checked for collisions and other safety constraints, and subsequently executed on-robot. The user then repeats this process until the task is complete.

Note that we use the same data pipeline and motion primitives when executing our Transporter Networks model as is used during human teleoperation; the human UI interaction is replaced with our inference results, and additional logic is added to switch between assembly and disassembly tasks. Autonomous mode is executing within the standard UI by first inputting the task to execute (by unique task code identifier), and pressing start. The UI for this (and visualization of inference results) is shown in Fig 14.

Using the above system we collected approximately 8,141 pick-and-place actions from 13 human operators in order to train the kit assembly model, an additional 2,300 actions for validation of model performance and hyper parameter tuning, and 6,759 pushing actions to train the sweeping model.

Success Detection

For the on-robot evaluation in Section 4.2, we manually label robot actions for success or failure. However this process is time consuming and expensive. Additionally, our Transformer Network does not estimate when a multi-action kitting episode terminates (either successfully or unsuccessfully). So in order to perform fully-autonomous pick-and-place without manual success labelling, an automatic task-level success signal was implemented. We trained a ResNet50 classification model - pretrained on imagenet classification -

to perform 3-way classification in order to detect full, empty or partial kits. We trained the classifier on 4848 training samples and evaluated on 128 validation set samples from a unique camera angle. Mean classification accuracy on the validation set was **96.9%**. Upon completion of a pick-and-place attempt, classifier inference on the RGB image produces a success metric, which marks the successful completion of a Kitting (or emptying the kit) episode. This allows the fully autonomous mode of the system to alternate between tasks (kitting and emptying) upon completion.