The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Deictic Image Mapping: An Abstraction for Learning Pose Invariant Manipulation Policies

Robert Platt, Colin Kohler, Marcus Gualtieri

College of Computer and Information Science, Northeastern University
360 Huntington Ave, Boston, MA 02115, USA
{rplatt,ckohler,mgualti}@ccs.neu.edu

Abstract

In applications of deep reinforcement learning to robotics, it is often the case that we want to learn pose invariant policies: policies that are invariant to changes in the position and orientation of objects in the world. For example, consider a peg-in-hole insertion task. If the agent learns to insert a peg into one hole, we would like that policy to generalize to holes presented in different poses. Unfortunately, this is a challenge using conventional methods. This paper proposes a novel state and action abstraction that is invariant to pose shifts called *deictic image maps* that can be used with deep reinforcement learning. We provide broad conditions under which optimal abstract policies are optimal for the underlying system. Finally, we show that the method can help solve challenging robotic manipulation problems.

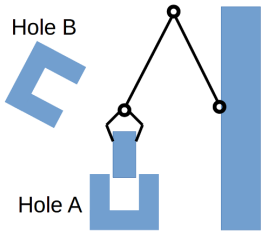


Figure 1: A policy for inserting a peg into Hole A learned using deictic image mapping generalizes to Hole B without any additional training.

Introduction

Policies learned by deep reinforcement learning agents are generally not invariant to changes in the position and orientation of the camera or objects in the environment. For example, consider the peg-in-hole insertion task shown in Figure 1. A policy that can insert the peg into *Hole A* does not necessarily generalize to *Hole B*. In order to learn a policy that can perform both insertions, the agent must be presented with examples of the hole in both configurations during training. This is a significant problem in robotics because we often want to learn policies most easily expressed relative to an object (i.e. relative to the hole) rather than relative to an image. Without the ability to generalize over pose, deep reinforcement learning must be trained with experiences that span the space of all possible pose variation. For example, in order to learn an insertion policy for any hole configuration, the agent must be trained with task instances for hole configurations spanning $SE(2)$, a three dimensional space. The problem is even worse in $SE(3)$ which spans six dimensions. The need for all this training data slows down learning and hampers the agent's ability to generalize over other dimensions of variation such as task and shape. While pose invariance is not always desirable, the inability to generalize over pose can be a major problem.

This paper introduces *deictic image mapping*, an approach to encoding robotic manipulation actions that generalizes over the pose of the robot and other objects in the world. We focus on the end-to-end learning setting in robotics where state is typically encoded in terms of images and the action set corresponds to possible motions of the robot. Our key idea is to encode action as an image of the world that is centered, aligned, and focused on the target pose of the motion. Since this representation encodes motion actions in terms of the configuration of the world relative to the endpoint of the motion, it generalizes well over changes in object pose and camera viewpoint. We call this a "deictic" representation because motion actions are encoded relative to the environment rather than an absolute reference frame.

One of the interesting aspects of this work is that we show that for a large class of problems, the proposed deictic state and action abstraction induces an MDP homomorphism (Ravindran 2004; Ravindran and Barto 2003). In consequence, we are assured that optimal policies found in using the deictic representation induce optimal policies for the original problem. An important limitation of our approach is the large action set that is often required: we have as many of 26.9k actions in our experiments. This paper introduces a number of techniques for handling this large branching factor in a Deep Q-learning (DQN) framework. Finally, we report on a series of experiments that evaluates the approach both in simulation and in hardware. The results show that the method can solve a variety of challenging manipulation problems with training times less than two hours on a standard desktop GPU system.

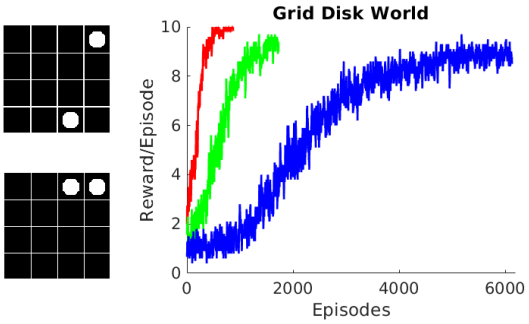


Figure 2: Left: the GRID-DISK domain. Disks are initially placed on a grid randomly (top left). The agent must pick one disk and place it adjacent to another (bottom left). Right: learning curves averaged over 10 runs for DQN applied to GRID-DISK for a 3×3 grid (red), a 4×4 grid (green), and a 5×5 grid (blue).

Problem Statement

We will consider problems expressed for a class of robotic systems that we call *move-effect* systems.

Definition 1 (Move-Effect System)**.** *A move-effect system is a discrete time system comprised of a set of effectors mounted on a fully actuated platform that operates in a Euclidean space of dimension $d \in \{2, 3\}$. On every time step, the system perceives an image or signed distance function, I, and then executes a collision-free motion, $a_m \in \mathcal{A}_M \subset SE(d)$, of the platform followed by a motion, $a_e \in \mathcal{A}_\mathcal{E}$, of the effectors.*

A good example of a move-effect robot is a robotic arm engaged in prehensile manipulation. The robot may move its hand (i.e. the "platform") to any desired reachable pose $a_m \in \mathcal{A}_M \subset SE(3)$ via a collision-free motion. Once there, it may execute an effector motion $a_e \in \mathcal{A}_E$ such as opening or closing the hand. More complex manipulation systems can also be formalized this way. For example, a domestic robot that navigates within a house performing simple tasks can be expressed as a move effect system in $SE(2)$ where the simple tasks to be performed are the "effector motions".

Formulation as a Markov decision process

A Markov decision process (an MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R)$ where $\mathcal{S}$ denotes a set of system states and $\mathcal{A}$ a discrete set of actions. $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition probability function where $T(s_t, a_t, s_{t+1})$ is the probability of transitioning to state s_{t+1} when action a_t is taken from state s_t. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the expected reward of executing action a from state s. The solution to an MDP is a control policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected sum of future discounted rewards when acting under the policy. We will assume that a motion planner (e.g. an RRT or trajectory optimization planner) is available that can find a collision free path to any reachable pose.

In order to use the MDP framework for move-effect systems, we need to define state and action sets. An action is defined to be a pair, $a = (a_m, a_e) \in \mathcal{A}$ where $\mathcal{A} = \mathcal{A}_M \times \mathcal{A}_E$.

The agent does not observe state directly, but it can observe an image $I \in [0 \dots 2^{16}]^{|grid|} = \mathcal{I}$, taken at the beginning of each time step taken via a camera and/or depth sensor. Here, $grid \subset \mathbb{R}^d$ denotes the positions of a finite set of points corresponding to the pixels or voxels in the image or signed distance function. The system also has access to the configuration of its effectors, $\theta \in \Theta$, obtained using joint position sensors. We define state to be a history of the k most recent observations and actions:

$$s_t = \langle I_t^{1-k}, \theta_t^{1-k}, a_t^{1-k}, \dots, I_t^{-1}, \theta_t^{-1}, a_t^{-1}, I_t, \theta_t \rangle, \quad (1)$$

where I_t and θ_t denote the current values for those two variables and I_t^{-i}, θ_t^{-i}, and a_t^{-i} denote the respective variables as they were i time steps in the past (i.e. i time steps prior to t). $\mathcal{S} = \mathcal{I}^k \times \Theta^k \times \mathcal{A}^{k-1}$ is the set of all states. History based representations of state as in Equation 1 are often used in deep reinforcement learning (e.g. (Mnih et al. 2013)).

Why move-effect problems are challenging

The standard MDP formulation described in the previous section is not well suited to move-effect systems. For example, consider the GRID-DISK domain as shown in Figure 2, where the argent must pick one disk and place it horizontally adjacent to the other. Transitions are deterministic: a pick (resp. place) succeeds if executed for an occupied (resp. unoccupied) cell and does not otherwise. This example is a move-effect system where $\mathcal{A}_M \subset SE(2)$ is the set of 16 grid positions and $\mathcal{A}_E$ contains exactly one pick and one place action (32 actions total). On each time step, the agent observes an image of the grid as well as a single bit that denotes the configuration of its hand – either open or closed (Equation 1 for $k = 1$). Using DQN on the MDP formulation described above, the number of episodes needed to learn a good policy increases as the number of cells in the grid increases, as shown in Figure 2, right. In this experiment, we used a standard ϵ-greedy DQN with dueling networks (Wang et al. 2015), no prioritized replay (Schaul et al. 2015), a buffer size of 10k, a batch size of 10, and an episode length of 10 steps. Epsilon decreased linearly from 100% to 10% over the training session. The neural network has two convolutional+relu+pooling layers of 16 and 32 units respectively with a stride of 3 followed by one fully connected layer with 48 units. We use the Adam optimizer with a learning rate of 0.0003.

Deictic image mapping

Move-effect systems have structure that makes problems involving these systems easier to solve than the results in Figure 2 suggest. Notice that in the case of GRID-DISK, the optimal policy is most easily expressed relative to current disk positions on the grid: the agent must learn to pick up a disk and place it horizontally or vertically adjacent to the other. This reflects a symmetry in the problem whereby world configurations where objects occupy the same relative configurations have similar (i.e. symmetric) transitions and reward outcomes. In order to improve learning performance, we need to encode the problem in a way that reflects this symmetry. Note that this cannot be accomplished just by switching to an actor critic method like DDPG (Lillicrap et al.

2015). DDPG could make it easier for the agent to learn to generalize over position, but it cannot generalize over orientation. Instead, we introduce deictic image state and action mappings that induce an abstract MDP that captures problem symmetries and can be solved using DQN.

Action mapping

As expressed in the Problem Statement, each action is a pair, $a_t = \langle a_m(t), a_e(t) \rangle \in \mathcal{A} = \mathcal{A}_M \times \mathcal{A}_E$, where we use the notation $a_m(t)$ to describe the destination of the platform motion and $a_e(t)$ to describe the effector motion at time t. Our key idea is to express a_m in terms of the appearance of the world in the vicinity of a_m rather than as coordinates in a geometeric space. Recall that on each time step, the move-effect system observes an image or signed distance function, $I \in \mathcal{I} = [0 \ldots 2^{16}]^{|grid|}$, expressed over a finite set of positions, $grid \subset \mathbb{R}^d$, corresponding to pixels or voxels. Let $crop(I, a_m) \subset \mathcal{I}$ denote a cropped region of I centered on a_m and aligned with the basis axes of a_m and let $\mathcal{I}'$ denote the space of all possible crops. The action mapping is then $g_s : \mathcal{A} \rightarrow \mathcal{A}' = \mathcal{I}' \times \mathcal{A}_E$ where:

$$g_{s_t}(a_t) = \langle crop(I_t, a_m(t)), a_e(t) \rangle, \tag{2}$$

I_t is an element of s_t, and $a_m(t)$ and $a_e(t)$ are elements of a_t. We call this a *deictic image* action mapping because it uses an image to encode the motion relative to other objects in the world. This can be viewed as an action abstraction (Ravindran 2004; Ravindran and Barto 2003) and we will call $\mathcal{A}'$ the *abstract action* set. This encoding works well for move-effect problems where the outcome of an effect action depends only on the local geometry of the world where the action was executed.

State mapping

The action mapping of Equation 2 induces a state abstraction. Recall that we defined state to be the history of the last k observations and actions (Equation 1). We can simplify this representation by using the action abstraction of Equation 2. The easiest way to do this is to define abstract state to be the current robot state paired with a history of the $k-1$ most recently executed abstract actions:

$$\begin{aligned} f_k(s_t) = \langle \{ \quad & crop(I_t^{1-k}, a_m^{1-k}(t)), a_e^{1-k}(t), \ldots, \\ & crop(I_t^{-1}, a_m^{-1}(t)), a_e^{-1}(t), \theta_t \rangle \end{aligned} \tag{3}$$

where I_t^{-i}, $a_m^{-i}(t)$, and $a_e^{-i}(t)$ are elements of s_t and $\mathcal{S}' = \mathcal{I}'^{k-1} \times \mathcal{A}_E^{k-1} \times \Theta$ is the set of abstract states. When k is understood, we will sometimes abbreviate $f = f_k$. We refer to $\mathcal{S}'$ as the *abstract state set*.

DQN in the abstract state and action space

The state and action mappings introduced above induce an abstract MDP that can be solved using DQN. Let $Q' : \mathcal{S}' \times \mathcal{A}' \rightarrow \mathbb{R}$ denote the abstract action-value function, expressed over the abstract state and action space, encoded by a deep neural network. At the beginning of each time step, we evaluate the abstract state, $s'_t = f(s_t)$, for the current state, s_t. We can calculate the greedy action for the current state s_t with respect to the abstract action-value function Q' using:

$$a^* = \arg\max_{a \in \mathcal{A}} Q'(f(s_t), g_{s_t}(a)). \tag{4}$$

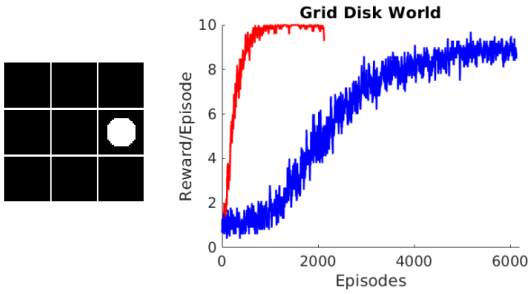


Figure 3: Left: example of a deictic action representation. Right: deictic (red) versus DQN baseline (blue) for 5×5 GRID-DISK.

After each time step, we store the underlying transition, (s_t, a_t, s_{t+1}, r_t) in the standard way. Training is nearly the same as usual except that after sampling a mini-batch, we calculate targets using the abstract state-action value function, $r_t + \max_{a \in \mathcal{A}} Q'(f(s_t), g_{s_t}(a))$, rather than in the standard way. The neural network that encodes Q' takes an abstract state-action pair as input and outputs a single estimate of value. In our experiments, we use a standard convolutional architecture comprised of two convolution+relu+pooling layers followed by fully connected layers.

Consider the following example of deictic image mapping for the GRID-DISK domain introduced earlier. Recall that for an image $I \in \mathcal{I}$ and a platform motion $a_m \in \mathcal{A}_M$, $crop(I, a_m)$ is a cropped region of I centered and aligned with a_m. $\mathcal{A}_M$ is the set of platform motion target locations corresponding to the cells ($|\mathcal{A}_M| = 16$ for 4×4 GRID-DISK). In this example, we define $crop(I, a_m)$ to be the 3×3 cell square region centered on a_m. (We use zero-padding for a_m positions on near the edge of the grid.) For example, Figure 3 left shows $crop(I, a_m)$ for the place action shown in Figure 2b. Since the agent is to place a disk to the left of an existing disk, the deictic image for this place action shows the existing disk to the right of an empty square where the place will occur. θ is the configuration of the effector: in this case just the configuration of the gripper jaws. Figure 3 right compares learning curves for DQN with deictic image states and actions versus the DQN baseline averaged over 10 runs each for the 5×5 GRID-DISK domain. DQN is parameterized just as it was in the last section. Notice that deictic image mapping speeds up learning considerably.

Scaling up

A key challenge for deictic image mapping is the large branching factor that is created by using end-to-end planned motions rather than small displacements as actions. In standard DQN, we must evaluate the Q-value of each action in order to select one for execution. However, many problems of practical interest involve tens or hundreds of thousands of

potential actions. One approach to handling this would be to adapt an actor critic method such as DDPG to the deictic image mapping setting. However, we have so far only explored the DQN version of our approach and leave an actor critic version for future work. Instead, this section introduces several techniques that together enable us to handle the large branching factor for problems in SE(2). Note that large action spaces are becoming more common in the literature. For example, (Li, Hsu, and Lee 2018) uses a hand-coded heuristic to prune action choices and (Zeng et al. 2018) uses a fully convolutional network architectures to evaluate Q-values for millions of actions.

Passing multiple actions as a batch to the Q-network: Perhaps the easiest way to handle the large number of actions is to pass a large set of action candidates to the Q-network as a single batch. Neural network back-ends such as TensorFlow are designed for this and it enables us to evaluate as many as 4.6k actions in a single forward pass on a standard NVIDIA 1080 GPU.

Keeping an estimate of the state-value function: The large action branching factor is a problem every time it is necessary to evaluate $\max_{a \in \mathcal{A}} Q'(f(s_t), g_{s_t}(a))$. This happens twice in DQN: once during action selection and again during training when evaluating target values for a mini-batch. We eliminate the second evaluation by estimating the abstract state-value function, V', concurrently with the abstract action-value function, Q', using a separate neural network. V' is trained using targets obtained during action selection: each time we select an action for a given state, we update the V' network using the just-evaluated max for that state. This enables us to calculate a target using $r_t + \gamma V'(f(s_{t+1}))$ rather than $r_t + \gamma \max_{a \in \mathcal{A}} Q'(f(s_t), g_{s_t}(a))$.

Value function hierarchy: Another way to speed up evaluation of $\max_{a \in \mathcal{A}} Q'(f(s_t), g_{s_t}(a))$ is to leverage a hierarchy of value functions specifically designed for move-effect problems in $SE(2)$ (Gualtieri and Platt 2018). Rather than estimating Q' directly using a single neural network, we use a hierarchy of two networks, Q'_1 and Q'_2 (although one could envision additional levels of hierarchy). Q'_2 is trained in the standard way: for each s, a pair in a mini-batch, $Q'_2(f(s), g_s(a))$ is trained with the corresponding target. However, we also train $Q'_1(f(s), g_s(Fix(a)))$ with the same target, where $Fix(a) \in SE(2)$ denotes the pose with the same position as a but with an orientation that is fixed with respect to the platform reference frame. Essentially, the Q'_1 estimate is an average of the value function over all orientations for each given position.

Instead of maximizing by evaluating Q'_2 for the position and orientation of all possible motion actions, we score each position using Q'_1 and then maximize Q'_2 over the positions and orientations corresponding to the top scoring positions under Q'_1. This is essentially a graduated non-convexity method (Blake and Zisserman 1987). First, we evaluate $Q'_1(f(s), g_s(\bar{a}))$ over all positions, $\bar{a} \in \bar{\mathcal{A}} = \{Fix(a) : a \in \mathcal{A}\}$. Let $\bar{\mathcal{A}}_{top}$ denote the top scoring η percent of the abstract actions in $\bar{\mathcal{A}}$. Then, we evaluate $\max_{a \in \bar{\mathcal{A}}_{top}} Q'_2(f(s), g_s(Fix^{-1}(a)))$, where Fix^{-1} denotes the one-to-many inverse of Fix that returns all fully-specified poses that correspond to positions in $\bar{\mathcal{A}}_{top}$. This approach enables us to estimate the maximum of Q'_2 without exhaustively evaluating the function for all poses in $SE(2)$. Like all graduated non-convexity methods, this approach is not guaranteed to provide an optimum. However, our results indicate that it works reasonably well on our problems (see the comparison in the next section).

Using hand-coded heuristics: Another way to reduce the action branching factor is to encode human knowledge about which kinds of motions are likely to be relevant to the problem. This type of approach has been used by others including (Li, Hsu, and Lee 2018) who hand-coded a heuristic for identifying which push actions. In this paper, we constrain the system to move only to positions that are within a neighborhood of some other visible object: we discard all motions, a_m, where the associated cropped image patch, $crop(I, a_m)$, does not include pixels with some positive height above the table plane.

Curriculum learning: Another thing that can speed up training is curriculum learning (Bengio et al. 2009). This basically amounts to training the agent to solve a sequence of progressively more challenging tasks. The hard part is defining the task sequence in such a way that each task builds upon knowledge learned in the last. However, this is particularly easy with deictic image mapping. In this case, we just need to vary the discretization of $\mathcal{A}_M$ on each curriculum task, starting with a coarse and ending with a fine discretization. For example, suppose we want to learn a GRID-DISK policy over a fine grid of possible disk locations. We would initially train using a coarse discretization and subsequently train on a more fine discretization. Although some of the image patches in the fine discretization will be different from what was experienced in the coarse discretization, these new image patches will have a similar appearance to nearby patches in the coarse discretization. Essentially, platform motions that move to similar locations will be representated by similar image patches. As a result, policy knowledge learned at a coarse level will generalize appropriately at the fine level.

Experiments

We evaluate deictic image mapping in simulation and on a robot for problems with an action set that spans $SE(2)$.

Block alignment in simulation

We evaluate on the block alignment problem shown on the left side of Figure 4 where the agent must grasp one block and place it in alignment with the other. We use a dense discretization of $SE(2)$ in this problem: the agent can choose to perform either a pick or place action (2 possibilities) at any position on a 29×29 grid (841 possibilities) at one of 16 orientations over 180 deg (equivalent to 32 orientations over 360 deg because of gripper symmetry), for a total of 26.9k different actions (last column of Table 1). We perform experiments to answer the following questions. (1) Can deictic image mapping solve the block alignment problem and how does it perform relative to a flat DQN solution? (2) How much does curriculum learning help? (3) What effect does the value function hierarchy have on learning?

Curriculum stage number	1	2	3	4	5	6	7	8
Object Type	Disks	Disks	Blocks	Blocks	Blocks	Blocks	Blocks	Blocks
Num Positions	25	25	25	25	81	289	841	841
Num Orientations	2	8	2	4	8	8	8	16
Num Actions	100	400	100	200	1.2k	4.6k	13.5k	26.9k

Table 1: Eight-stage curriculum used in the simulated experiments.

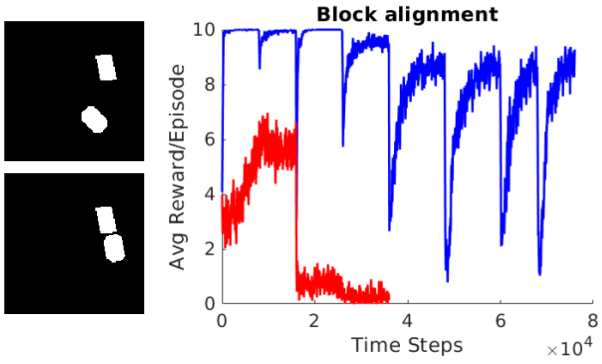
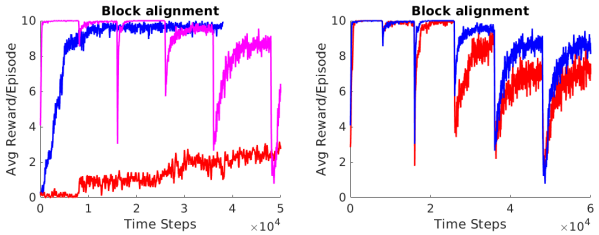


Figure 4: Left: block alignment problem. Top left: two blocks are placed randomly. Bottom left: the agent must pick up one block and place it roughly in alignment with the second. Right: learning curves for simulated blocks world domain. Blue: DIM with hierarchical value function only in last two curriculum steps. Red: DQN baseline.



(a) (b)

Figure 5: (a) Ablation of curriculum learning. Blue, red: learning curves for curriculum stages 4 and 5, respectively, without running stages 1–3 first. Magenta: learning curve with curriculum. (b) Ablation of hierarchical value function. Blue: WITHOUT Q-function hierarchy; Red: WITH Q-function hierarchy.

(1) Comparison with a DQN baseline: Here, we follow the eight-task curriculum shown in Table 1, starting with a task involving disks as in Figure 2 and ending with the block alignment task. During the first six stages of curriculum learning, we use all of the scaling-up techniques described in the last section except for the value function hierarchy (as we show later, this slows learning performance). We only turn on the value function hierarchy in the last two stages because the NVIDIA 1080 ran out of memory without the hierarchy. Both DQN and deictic image mapping was parameterized exactly as it was earlier in the paper except that the deictic experiments used prioritized replay with $\alpha = 0.6$, $\beta = 0.4$, and $\epsilon = 0.000001$ and we start ϵ-greedy exploration with $\epsilon = 0.5$ instead of $\epsilon = 1$ (but ϵ still decreases linearly with time). The full training curriculum executes in approximately 1.5 hours on a standard Intel Core i7-4790K running one NVIDIA 1080 graphics card.

Figure 4 shows the results. DQN performance is shown in red while deictic image mapping performance is shown in blue. Notice the "spikes" downward in the learning curve. Each spike corresponds to the start of a new task in the curriculum (a total of eight spikes including the one at episode zero). After the agent solves a task in the curriculum, this triggers a switch to a new task. Performance drops while the agent learns the new task but then recovers. The DQN agent was stopped after the fourth curriculum stage because average reward per episode had dropped to nearly zero. Notice that DQN completely fails to solve this task – it only learns anything meaningful in the easiest stages of the curriculum. In contrast, the deictic agent does well throughout all eight curriculum stages. Note that average reward per episode is slightly lower on the fourth curriculum stage and onward because these tasks involve blocks in at least four orientations which can sometimes be infeasible. This occurs when both blocks are placed diagonally in a corner so that it is impossible to pick up either block and place it in alignment with the other.

(2) The effect of curriculum learning: We investigate learning performance if we were to start learning for stages 4 and 5 of the curriculum without first going through the prior curriculum steps. Here, the neural network is initialized with random weights and we allow the agent to learn for the same number of time steps as would be required by the curriculum to get to that point. Figure 5a shows the results. The agent is able to learn stage 4 of the curriculum relatively quickly starting from random weights, but it cannot learn stage 5. This suggests that curriculum stages 1–3 are superfluous but that stages 5 and onward cannot be learned without some form of curriculum learning. Looking at Table 1, this makes sense because it is at stage 5 that the number of actions jumps from 400 to 1.2k.

(3) The effect of the value function hierarchy: Another part of our strategy for speeding up the Q-function maximization operation is the value function hierarchy strategy outlined earlier. We compare learning performance for the first six curriculum stages with and without this additional feature. Recall that η is the percentage of top scoring positions in Q'_1 that are selected to further evaluation in Q'_2. A larger η results in faster evaluation, but a potentially worse

Task	Success Rate	Avg # Steps	Num Trials
3-in-row cube	94%	7.8	50
3 Cube Stack	90%	4.7	50
4 Cube Stack	84%	6.9	50
2 Rectangular Block Stack	80%	N/A	83
Cube in mug	94%	2.72	50

Table 2: Results from robot experiments.

estimate of the maximum. Here, we set $\eta = 0.2$ Figure 5b shows the comparison. Notice that the average reward per episode is approximately 15% worse with the hierarchy as compared to without it. We conclude that while the value function hierarchy is sometimes needed to train in a reasonable period of time, it should be avoided if possible.

Robot experiments

We performed robot experiments to get a sense of the kinds of tasks that deictic image mapping can help solve. We used a UR5 equipped with a Robotiq two finger gripper, as shown in Figure 6a. In all experiments, objects were dumped randomly onto a table in randomly selected positions within an 50×50 cm region within the manipulator workspace. All end-to-end motions commanded by the agent were planned using TrajOpt (Schulman et al. 2013) with the final gripper pose constrained to be orthogonal to the table. All images were acquired using a depth sensor (Structure IO, the light blue device in Figure 6a,b) mounted to the gripper. At the beginning of each time step, because of camera calibration inaccuracies, the robot took images from four different positions above the table and stitched them together to create a complete 2-D image. Then, the system selected and performed a pick or place action consisting of a single collision free motion followed by an opening or closing motion of the gripper. In all experiments, the system was trained in completely in simulation using OpenRAVE (Diankov and Kuffner 2008) using a simulated setup nearly identical to the physical robot. Then, the policy learned in simulation was executed and evaluated on the real robot. We evaluated performance for the following five tasks. All tasks had sparse rewards: the agent received 0 reward everywhere except upon reaching the goal state.

Three-in-a-row cube arrangement: In this task, three cubes were dumped onto a table and then re-oriented manually so that each block had a fixed orientation with respect to the robot. The system was trained using the same parameters as used in simulation, but using four curriculum stages starting with a 10cm step size and annealing down to a 1cm step size. The agent received +1 reward for achieving a three-in-a-row alignment within ten time steps and zero otherwise. As shown in the first row of Table 2, we obtain a 94% task success rate over 50 trials.

Stacking three cubes: This task is the same as three-in-a-row (above) except that the agent now received +1 reward for stacking the three cubes. We obtain a 90% task success rate (second row of Table 2).

Stacking four cubes: This task is the same as above except that the agent must now stack four cubes (Figure 6b). We obtain an 84% task success rate (third row of Table 2).

Stacking two rectangular blocks: In this task, eight rectangular blocks were dumped on the table (Figure 6c). The robot received +1 reward every time it stacked one block on to of another (Figure 6d). After every successful stack, the top block was removed from the table by the experimenter. Unlike the three cube stacking/arranging tasks above, the action space for this task included 16 possible orientations (similar to the simulation experiments). The agent was trained using a seven stage curriculum and the value function hierarchy with $\eta = 0.1$ that reduced the number of actions from 51.2k down to 8.32k. This task is interesting because the agent must learn to balance long blocks on top of small blocks as shown in Figure 6d. The average success rate for two-block stacking attempts was 80% (fourth row of Table 2).

Cube in mug: Finally, we evaluated our approach on a task where the robot must pick up a cube and drop it into a mug (Figure 6f). During robot testing, at the start of each episode, a random mug was chosen from the selection of eight mugs shown in Figure 6e and was randomly placed on the table in a upright position along with a block. Importantly, these were novel mugs in the sense that they were different from any of the six randomly re-scaled mugs used to train the system. The agent received a reward of -0.1 when it moved the mug and +1 when it solved the task within no more than 10 time steps. The average task success rate here was 94% (last row in Table 2).

These five robot experiments suggest that deictic image maps can be used to solve a variety of robotic manipulation problems related to pick and place. The stacking-four-cubes experiment demonstrates that the approach can solve problems that involve several pick/place actions. The stacking-two-rectangular blocks experiments shows that the method can perform relatively precise placements. Finally, the cube-in-mug experiment shows that the method can be trained to solve tasks involving novel objects.

Analysis

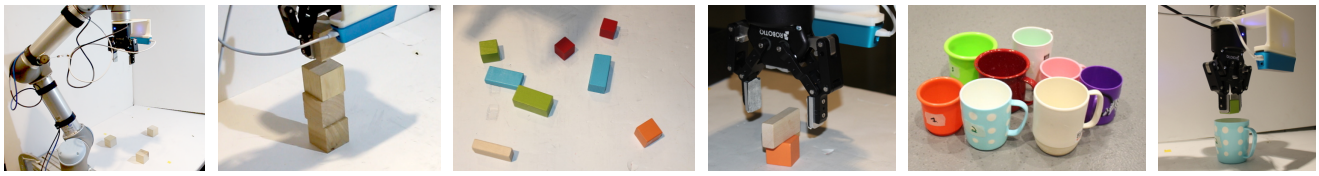
It turns out that deictic image mapping is theoretically correct in the sense that optimal policies for the abstract MDP found using DQN induce optimal policies for the original system. We use the MDP homomorphism framework.

Definition 2 (MDP Homomorphism, Ravindran 2004 (Ravindran 2004))**.** *An MDP homomorphism from an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ onto $\mathcal{M}' = \langle \mathcal{S}', \mathcal{A}', T', R' \rangle$ is a tuple of mappings, $\langle f, \{g_s | s \in \mathcal{S}\} \rangle$, where $f : \mathcal{S} \to \mathcal{S}'$ is a state mapping, $g_s : \mathcal{A} \to \mathcal{A}', s \in \mathcal{S}$ is a set of action mappings, and the following two conditions are satisfied for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $s' \in \mathcal{S}'$:*

$$T'(f(s), g_s(a), s') = \sum_{\bar{s} \in \{s \in \mathcal{S} | f(s) = s'\}} T(s, a, \bar{s}) \quad (5)$$

$$R'(f(s), g_s(a)) = R(s, a). \quad (6)$$

$\mathcal{S}'$ and $\mathcal{A}'$ are the *abstract* state and action sets, respectively. Recall that the action mapping introduced in Equa-



(a) (b) (c) (d) (e) (f)

Figure 6: (a) Experimental scenario. (b) four block stacking task. (c) initial block configuration for rectangular block stacking. (d) rectangular block stacking task. (d) Mugs used in cube-in-mug task. (f) Cube-in-mug task.

tion 2 encodes an action a in terms of $crop(I, a_m)$. Since I is the image or signed distance function perceived by sensors, this is a *state-dependent* mapping. The MDP Homomorphism framework is specifically relevant to this situation because, unlike other abstraction frameworks (Givan, Dean, and Greig 2003; Dean, Givan, and Leach 1997), it allows for state dependent action abstraction. If we can identify conditions under which the action mapping of Equation 2 in conjunction with the state mapping of Equation 3 is an MDP homomorphism, then a theorem exists that says that a solution to the abstract MDP induces a solution in the original problem:

Theorem 1 (Optimal value equivalence, Ravindran 2004 (Ravindran 2004)). *Let $\mathcal{M}' = \langle \mathcal{S}', \mathcal{A}', T', R' \rangle$ be the homomorphic image of $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ under the MDP homomorphism, $\langle f, \{g_s | s \in \mathcal{S}\} \rangle$. For any $(s, a) \in \mathcal{S} \times \mathcal{A}$, $Q^*(s, a) = Q'^*(f(s), g_s(a))$.*

Here, $Q^*(\cdot)$ and $Q'^*(\cdot)$ denote the optimal action-value function for the underlying MDP and the abstract MDP respectively. We now show that the deictic image mapping induces an MDP homomorphism.

Theorem 2. *Given a move-effect system with state set $\mathcal{S}$; action set $\mathcal{A}$; transition function T such that for all $t \in \mathbb{R}_{>0}$, θ_{t+1} is conditionally independent of s_t given $crop(I_t, a_m(t))$; a deictic image mapping $\langle f, \{g_s | s \in \mathcal{S}\} \rangle$; and an abstract reward function $R' : \mathcal{S}' \times \mathcal{A}' \to \mathbb{R}$; then there exist a reward function R and an abstract transition function T' for which $\langle f, \{g_s | s \in \mathcal{S}\} \rangle$ is an MDP homomorphism from $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ to $\mathcal{M}' = \langle \mathcal{S}', \mathcal{A}', T', R' \rangle$.*

Proof. See Appendix 1 in supplementary materials. □

As a consequence of Theorem 2 and 1, we know that optimal solutions to the abstract MDP induced by Equations 3 and 2 induce optimal solutions to the original move-effect system as long as the conditions of Theorem 2 are satisfied.

Related Work, Limitations, and Discussion

This work is related to prior applications of deixis in AI. Evidence exists that the human brain leverages deictic representations during performance of motor tasks (Ballard et al. 1997). Whitehead and Ballard proposed an application of deixis to reinforcement learning, focusing on blocks world applications in particular (Whitehead and Ballard 1991). While our representations are only loosely related to Ballard's computational architecture, our work is closely related to the psychological evidence in (Ballard et al. 1997).

The analysis section of this paper leverages the MDP Homomorphism framework introduced by Ravindran and Barto (Ravindran 2004; Ravindran and Barto 2003). MDP Homomorphisms are related to other MDP abstraction methods (Givan, Dean, and Greig 2003; Dean, Givan, and Leach 1997). However, unlike those methods, the MDP Homomorphisms allow for the state-dependent action abstraction, a critical part of our proposed abstraction.

This paper is also generally related to a variety of recent works exploring deep learning for robotic manipulation. The work of (Zeng et al. 2018) is related to ours in that they estimate a Q-function over a large number of grasping and pushing actions, in their case by creating a fully convolutional Q-network. The work of (Li, Hsu, and Lee 2018) also involves a large action space of push actions, pruned using a heuristic. Other pieces of recent related manipulation work are (Fang et al. 2018) and (Gualtieri, ten Pas, and Platt 2018) who each propose methods of learning task-relevant grasps in the context of a reinforcement learning problem. Our work can be loosely viewed as an extension of recent grasp detection work by (Mahler et al. 2017) and (Gualtieri et al. 2016), however the focus here is on more complex manipulation tasks rather than just grasping.

In contrast to some of the work cited above, a nice aspect of our method is that it can be applied to a relatively general class of problems, such as those explored in our experiments. However, the method has important limitations. First, limits on the number of actions that can be realistically considered on a given time step means that it will be important to find ways to limit the number of motion actions that need to be considered. Another limitation is that the deictic image patch used to represent a motion only "sees" a portion of the total image. As a result, our method is not well suited to respond to non-local state information. Although we have not yet tried this, we expect that this problem can be solved by using a "foveated" action patch that "sees" the world around a potential action choice at multiple resolutions.

Acknowledgements

This work has been supported in part by the National Science Foundation through IIS-1426968, IIS-1427081, IIS-1724191, IIS-1724257, IIS-1763469, and IIS-1830425, NASA through NNX13AQ85G, Amazon through an ARA, and Google through an FRA.

References

Ballard, D. H.; Hayhoe, M. M.; Pook, P. K.; and Rao, R. P. 1997. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences* 20(4):723–742.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Blake, A., and Zisserman, A. 1987. *Visual reconstruction*. MIT press.

Dean, T.; Givan, R.; and Leach, S. 1997. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, 124–131. Morgan Kaufmann Publishers Inc.

Diankov, R., and Kuffner, J. 2008. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA.

Fang, K.; Zhu, Y.; Garg, A.; Kurenkov, A.; Mehta, V.; Fei-Fei, L.; and Savarese, S. 2018. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *arXiv preprint arXiv:1806.09266*.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence* 147(1-2):163–223.

Gualtieri, M., and Platt, R. 2018. Learning 6-DoF grasping and pick-place using attention focus. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, 477–486. PMLR.

Gualtieri, M.; ten Pas, A.; Saenko, K.; and Platt, R. 2016. High precision grasp pose detection in dense clutter. In *IEEE Int'l Conf. on Intelligent Robots and Systems*.

Gualtieri, M.; ten Pas, A.; and Platt, R. 2018. Pick and place without geometric object models. In *IEEE Int'l Conf. on Robotics and Automation (ICRA)*. IEEE.

Li, J. K.; Hsu, D.; and Lee, W. S. 2018. Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems (RSS)*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mahler, J.; Liang, J.; Niyaz, S.; Laskey, M.; Doan, R.; Liu, X.; Ojea, J. A.; and Goldberg, K. 2017. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Ravindran, B., and Barto, A. 2003. Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1011–1016.

Ravindran, B. 2004. *An algebraic approach to abstraction in reinforcement learning*. Ph.D. Dissertation, University of Massachusetts at Amherst.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J.; Ho, J.; Lee, A.; Awwal, I.; Bradlow, H.; and Abbeel, P. 2013. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*. Citeseer.

Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; and De Freitas, N. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Whitehead, S. D., and Ballard, D. H. 1991. Learning to perceive and act by trial and error. *Machine Learning* 7(1):45–83.

Zeng, A.; Song, S.; Welker, S.; Lee, J.; Rodriguez, A.; and Funkhouser, T. 2018. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *arXiv preprint arXiv:1803.09956*.