Spatial Action Maps for Mobile Manipulation

Jimmy Wu[1], Xingyuan Sun[1], Andy Zeng[2], Shuran Song[3],
Johnny Lee[2], Szymon Rusinkiewicz[1], Thomas Funkhouser[1,2]
[1]Princeton University   [2]Google   [3]Columbia University

Abstract—Typical end-to-end formulations for learning robotic navigation involve predicting a small set of steering command actions (e.g., step forward, turn left, turn right, etc.) from images of the current state (e.g., a bird's-eye view of a SLAM reconstruction). Instead, we show that it can be advantageous to learn with dense action representations defined in the same domain as the state. In this work, we present "spatial action maps," in which the set of possible actions is represented by a pixel map (aligned with the input image of the current state), where each pixel represents a local navigational endpoint at the corresponding scene location. Using ConvNets to infer spatial action maps from state images, action predictions are thereby spatially anchored on local visual features in the scene, enabling significantly faster learning of complex behaviors for mobile manipulation tasks with reinforcement learning. In our experiments, we task a robot with pushing objects to a goal location, and find that policies learned with spatial action maps achieve much better performance than traditional alternatives.

I. INTRODUCTION

The prolific success of deep learning in vision [16, 33, 13, 14] has inspired much recent work on using deep convolutional networks (ConvNets) for visual navigation and mobile manipulation in robotics. In such an approach, ConvNets are trained to model a policy that maps from an agent's observation of the state (*e.g.*, images) to the probability distribution over actions (*e.g.*, steering commands) maximizing expected task success.

Most work in this domain has focused on methods to train the ConvNets: from learning to imitate human control strategies [22, 4, 11, 24], to discovering more complex long-term path planning behaviors with reinforcement learning [20, 47, 7, 29]. Other works have considered the impact of different state representations: from forward-facing camera images [22, 26, 4, 24] to top-down bird's-eye views (BEV) of the scene generated with Inverse Perspective Mapping (IPM) [11, 7, 29, 5]. However, there has been very little work on the impact of different action representations. Almost all navigation systems based on ConvNets consider only a small set of (possibly parameterized) egocentric steering commands (*e.g.*, move forward/backward, strafe left/right, rotate left/right, etc.) [26, 20, 11, 12, 24, 47, 29, 7].

Representing actions with steering commands (Fig. 2a) presents several problems for learning complex mobile manipulation tasks. First, they are myopic, and thus each action can reach only a small subset of possible endpoints (*i.e.*, ones at a fixed distance and reachable by unobstructed straight-line paths). Second, they each invoke only a small change to the state, and thus a long sequence of actions is required to make significant change (*e.g.*, navigate through a series of obstacles). Third, they require a deep Q-network to learn a complex mapping from a high-dimensional input state representation
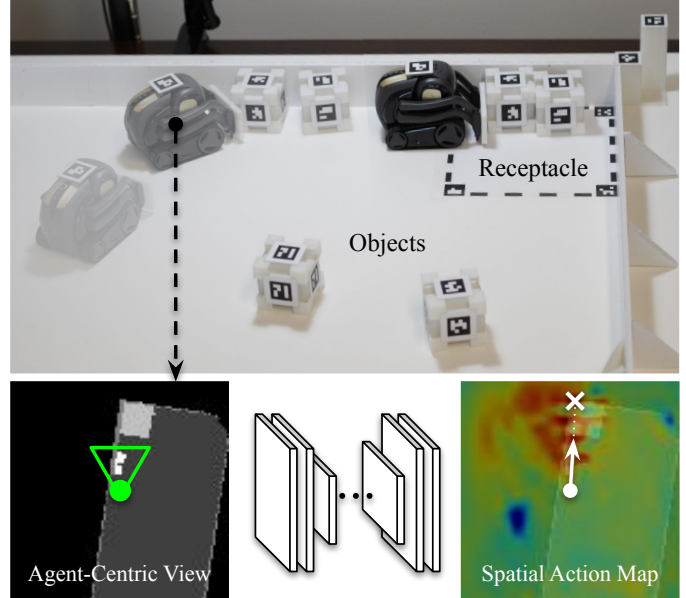

Fig. 1. In this work, we propose "spatial action maps" as an action representation for mobile manipulation tasks. In our setting, the goal is to train an agent to navigate around an unseen environment, find objects, and push them into a designated target receptacle. It does so by iteratively predicting the Q-value of navigating to each location in a spatial action map (red is higher), and then selecting the best location to go to (marked with 'x').

(usually an image) to a low-dimensional set of (possibly parameterized) action classes, which may require many training examples.

In this work, we advocate for a new action representation, "spatial action maps" (Fig. 1). The main idea is to represent actions as a dense map of navigation endpoints: each action represents a move to an endpoint, possibly along a non-linear trajectory, and possibly with a task to perform there (Fig. 2b-c). The advantages of this action representation are three-fold. First, the agent is not myopic – it can move to any endpoint in the spatial action map with a single action [1], enabling goal-driven behaviors. Second, each action can represent an arbitrarily complex navigation trajectory – ours follow shortest paths to endpoints while avoiding obstacles (Fig. 2c). Third, it simplifies the mapping from states to actions – in our system, each state is represented by an image of the reconstructed scene from a bird's-eye view (in IPM coordinates), and the action space is represented by an image representing the expected Q-value of navigating to every endpoint (also in IPM coordinates). Since the state and action space lie in the same domain and are pixel-aligned, we can train a fully convolutional network (FCN) to map between them – an efficient way to predict values for all possible actions in one forward pass of a network.

We study this action representation in the context of reinforcement learning for mobile manipulation, where an agent is tasked with exploring an unseen environment containing obstacles and objects, with the goal of navigating to push all objects into a designated target zone – *i.e.*, pushing the objects into a receptacle, like a bulldozer (Fig. 1). At every step, the only information available to the agent is an overhead image representing a partial reconstruction of its local environment (everything it has observed with a forward-facing RGB-D camera, transformed into an IPM representation and accumulated over time to simulate online SLAM/reconstruction). The agent feeds the state image into an FCN to produce an action image, which encodes the Q-value of moving to every endpoint location along the estimated shortest path trajectory. It executes the action with highest Q-value and iterates.

The spatial action map representation has the key benefit that the spatial position of each state-action value prediction (with respect to the input IPM view) represents a local milestone (trajectory endpoint) for the agent's control strategy. We conjecture that it is easier for convolutional networks to learn the state-action values of these navigational endpoints (as opposed to abstract low-level steering commands), since each prediction is aligned and translationally anchored to the visual features of the objects and obstacles directly observed in the input map. This is motivated by gains in performance observed in other domains – for example, image segmentation [14] has been shown to benefit from pixel-aligned input and output representations, while end-to-end robotic manipulation [21, 45, 44, 43] is significantly more sample efficient when using FCNs to predict state-action values for a dense sampling of grasps aligned with the visual input. Using spatial action maps, our experiments show that we are capable of training end-to-end mobile manipulation policies that generalize to new environments with fewer than 60k training samples (state-action pairs). This is orders of magnitude less data than prior work [20, 47].

Our main contribution in this work is the spatial action map representation for mobile manipulation. We investigate its use with a variety of action primitives (take a short step, follow the straight-line path, follow the shortest path) and state input channels (partial scene reconstructions, shortest path distances from the agent, shortest path distances to the receptacle). In simulation environments, we conduct ablation studies and find that our proposed state and action representations give better performance and learning efficiency compared to alternative representations. We show that our policies also work in real environments. Supplemental materials, including code, simulation environments, and videos of our robot in action, are available at https://spatial-action-maps.cs.princeton.edu.

II. Related Work

Navigation. There has been significant recent work on training agents to navigate virtual environments using first-person visuomotor control [15, 19, 27, 2, 37, 38, 39, 28]. In a typical setup, the agent iteratively receives first-person observations as input (*e.g.*, images with color, depth, normals, semantic

(a) Steering commands (b) Spatial action map (straight-line paths) (c) Spatial action map (shortest paths)
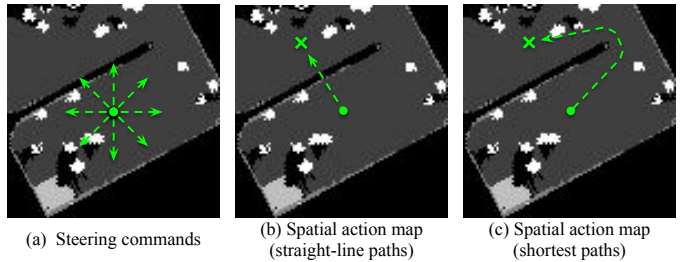
Fig. 2. **Action representations.** (a) shows all trajectories (green) available in a discrete set of steering commands. The other images show example trajectories available in spatial action maps (every pixel is a potential trajectory endpoint). We consider two variants: (b) where the agent follows a straight-line path to the selected target endpoint (denoted by 'x'), and (c) where it follows an estimated shortest path (our method).

segmentations, etc.), from which it builds a persistent state (*e.g.*, a map), and selects one of many possible actions as output (*e.g.*, move, rotate, strafe, etc.) until it completes a task (*e.g.*, navigate to a given location, find a particular type of object, etc.). These works focus almost exclusively on how to best train neural networks for the task, for example using deep reinforcement learning [19, 47, 6], supervised [34, 12] and self-supervised learning [25], or predicting the future [9]. They do not study how different parameterizations of network outputs (actions) affect the learning performance – *i.e.*, the inputs are always in one domain (*e.g.*, images, GPS coordinates, etc.) and the outputs are in another (*e.g.*, move forward, rotate right, interact, etc.). In contrast, we investigate the advantages of dense predictions using spatial action maps, where the inputs and outputs are represented in the same spatial domain.

Mobile Manipulation. While many navigation works assume a static environment, other works also consider environments with movable objects [32, 35, 18]. In these works, the agent navigates to a goal location by pushing aside movable obstacles that are in the way. For example, Stilman and Kuffner [31] propose the task of *Navigation Among Movable Obstacles*, where the agent navigates in an environment that contains both static structures (*e.g.*, walls and columns) and movable obstacles. While these methods do not assume a static environment, the task is similar – the agent only needs to navigate to the goal location. In contrast, in our task, the agent must come up with a navigational plan that will push all objects in the environment to the goal location, which is a much more complex problem.

Another line of work studies navigation for object manipulation [10, 23, 42, 17]. These tasks (*e.g.*, picking and rearrangement of objects) are similar to ours, but the robot-object interactions considered in these tasks (*e.g.*, grasping) are often more predictable and happen over short time horizons, which makes it possible to apply simple heuristic-based algorithms to separately handle navigation and interaction. In contrast, our setup requires long time-horizon robot-object interactions, which are less predictable and more difficult to plan.

Dense Action Representations. Our work is inspired by recent work on dense affordance prediction for bin picking. For example, the multi-affordance picking strategy of Zeng
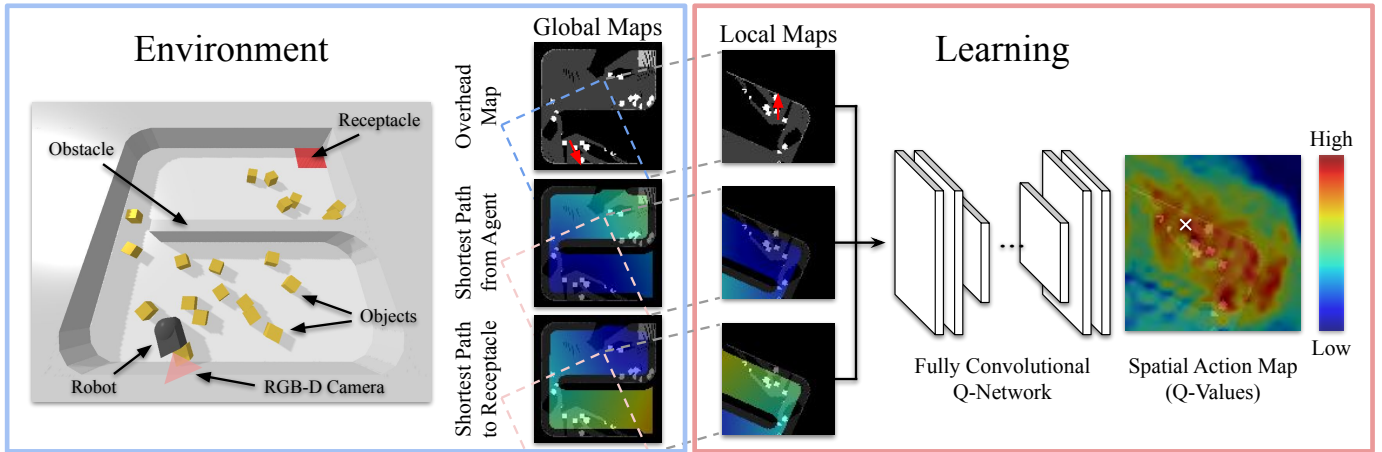
Fig. 3. **Overview.** We train in a simulation environment, where we mimic online SLAM/reconstruction as the agent moves around. This is implemented via a virtual forward-facing RGB-D camera mounted on the robot. The robot incrementally builds up a global overhead map of the room, as well as an occupancy map used to compute single-source shortest path distances. Locally oriented and aligned crops of these maps are used to train our fully convolutional Q-network, which produces a dense, pixel-aligned spatial action map (Q-value map). Note that the robot position channel (see Fig. 4) is omitted for clarity.

et al. [45] selects grasps by predicting a score for every pixel of an overhead view of a bin. Similar approaches are used to predict affordances for grasping or suction in [44, 46, 30, 41]. However, these systems are trained in the more constrained setting of bin picking or assembly, where supervision is available for grasp success, and where motion trajectories to achieve selected grasps can be assumed to be viable and of equal cost. In our work, we apply dense prediction to a more challenging scenario, where different actions have different costs (and some may not even be viable), and where long-term planning is required to perform complex action sequences.

III. METHODS

In this paper, we propose a new dense action representation in which each pixel of a bird's-eye view image corresponds to the atomic action of navigating along the shortest path to the associated location in the scene.

To investigate this action representation, we consider a navigate-and-push setting in which an agent must explore an unseen environment, locate objects, and push them into a designated target "receptacle" (Fig. 3). This task may be seen as an abstraction of real-world tasks such as bulldozing debris or sweeping trash, and is sufficiently complex that it would be difficult to implement an effective hand-coded policy or learn a policy with traditional action representations.

Our agent is an Anki Vector, a low-cost mobile tracked robot approximately 10 cm in length, augmented with a 3D-printed bulldozer-like end-effector (Fig. 1). The objects to be pushed are 4.4 cm cubic blocks, and the receptacle is located in the top right corner of a 1 m by 0.5 m "room" with random partitions and obstacles. For ease of prototyping, we include fiducial markers on the robot and the objects, and track them with an overhead camera. Our setting though, is intended to represent what would be possible with onboard sensing and SLAM. Therefore, the only information made available to the agent is a simulation of what would be observed via an onboard forward-facing RGB-D camera with a 90° field of view, integrated over time with online mapping – *i.e.*, our

agents do not have access to ground truth global state. This means that the agent must learn to act with partial or outdated information and actively seek out unexplored parts of the environment.

Our agents are trained in a PyBullet simulation [8], where state observations are generated by rendering camera views of the environment. We then execute learned policies in the real world, where the fiducial markers are used to update the state of the simulator (*e.g.*, robot and object poses). Simulation enables us to train our agents over a wider range of environments than would be possible with physical robots, while sim-to-real mirroring enables us to evaluate the robustness and generalization of our policies to real-world dynamics.

In the following subsections, we provide details about our training setup and describe how the appropriate action representation proves crucial to improving the sample efficiency and generalizability of our policies.

A. Reinforcement Learning (DQN) Formulation

We formulate the navigate-and-push problem as a Markov decision process: given state s_t at time t, the agent chooses to execute an action a_t according to a policy $\pi(s_t)$, then transitions to a new state s_{t+1} and receives a reward r_t. The goal of reinforcement learning is to find an optimal policy π^* that selects actions maximizing total expected rewards $Q(s_t, a_t) = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$, *i.e.*, a γ-discounted sum over an infinite horizon of future returns from time t to ∞.

In this work, we investigate off-policy Q-learning [20] to train a greedy policy $\pi(s_t)$ that chooses actions by maximizing a parameterized Q-function $\arg\max_{a_t} Q_\theta(s_t, a_t)$ (*i.e.*, state-action value function), where θ denotes the weights of our neural network (whose architecture we describe in Sec. III-D). We train our agents using the double DQN learning objective [36]. Formally, at each training iteration i, our objective is to minimize:

$$\mathcal{L}_i = |r_t + \gamma Q_{\theta_i^-}(s_{t+1}, \arg\max_{a_{t+1}} Q_{\theta_i}(s_{t+1}, a_{t+1})) - Q_{\theta_i}(s_t, a_t)|$$
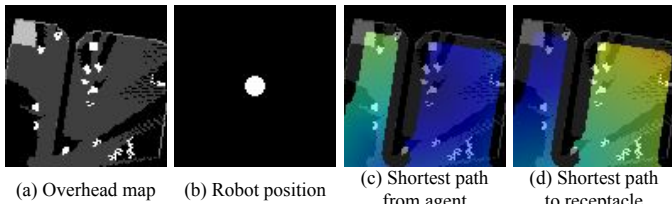
(a) Overhead map   (b) Robot position   (c) Shortest path from agent   (d) Shortest path to receptacle

Fig. 4. **Input channels.** From left to right, the pixel-aligned channels are all in the robot's local coordinate frame and correspond to (a) the observation, which is an overhead image of the reconstructed environment, (b) encoded robot position in local coordinates, which is always in the center of the image, (c) shortest path distance from the agent to each location in the image, and (d) shortest path distance to the receptacle from each location in the image. The receptacle is in the corner of the room, near the top left of these images. Color added for visualization only, green/yellow corresponds to higher values.

where (s_t, a_t, r_t, s_{t+1}) is a transition uniformly sampled from the replay buffer. Target network parameters θ^- are held fixed between individual updates. More training details are presented in Sec. III-D.

B. State Representation

Within our formulation, we represent the agent's observation of the state s_t as a visual 4-channel image from a local bird's-eye view that is spatially aligned and oriented with respect to the robot's local coordinate frame (such that the robot is positioned in the center of each image and looking along the y axis). This is similar to inverse perspective mapping (IPM), commonly used in autonomous driving [11, 7, 29, 5]. Each channel encodes useful information related to the environment [3] (visualized in Fig. 4) including: (1) a local observation of the robot's surroundings in the form of an overhead map, (2) a binary mask with a circle whose diameter and position encode the robot's respective size and location in the robot's coordinate frame, (3) an image where each pixel holds the shortest path distance from the agent to the corresponding location, and (4) an image where each pixel holds the shortest path distance to the receptacle from that location. The shortest path distances in the third and fourth channels are computed using an occupancy map generated only from local observations of obstacles (which are accumulated over time – see next paragraph) and normalized so that they contain relative values rather than absolute. All unobserved regions are treated as free space when computing shortest path distances. This reflects a realistic setting in which a robot has access to nothing but its own visual observations, GPS coordinates, local mapping, and task-related goal coordinates.

Online mapping. Online SLAM/reconstruction is a common component of any real mobile robot system. In our experiments, this is implemented in the simulation using images from a forward-facing RGB-D camera mounted on the robot. The camera captures a new image at the end of every transition. Using camera intrinsics and robot poses, depth data is projected into a 3D point cloud, then fused with previous views to generate and update a global map of the environment. At the beginning of each episode in a new environment, this global map is initially blank. As the robot moves around in the environment, the global map fills in as

(a) Environment   (b) Occupancy map   (c) Action trajectories
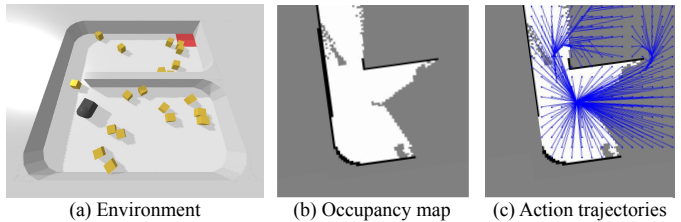
Fig. 5. **Action space.** In our action space, which is spatially aligned with the local state representation, every pixel represents a move to the corresponding navigational endpoint along the shortest path trajectory. The robot is located in the left of the environment (a) and in the center of local maps (b) and (c). Shortest path trajectories (c) are estimated using an occupancy map (b) built up with online mapping. Note that the action space is pixelwise dense, but we only show a subset of paths in (c) for clarity.

it accumulates more partial views over time. This restriction encourages the agent to learn a policy that can explore unseen areas in order to effectively complete the task. Specifically, the robot gradually reconstructs two global maps as it moves around: (1) a global overhead map, which encodes objects and obstacles, and (2) a global occupancy map, used for shortest path computations in our high-level motion primitives as well as our state representations and partial rewards. The agent makes no prior assumptions about the layout of obstacles in the environment.

C. Action Representation

Our actions are represented by an image (*i.e.*, action map, illustrated in Fig. 2) identically sized and spatially aligned with the input state representation. Each pixel in this action map corresponds to a navigational endpoint in local robot Cartesian coordinates. At each time step, the agent selects a pixel location in the action map – and therefore in the observed environment – of where it intends to move to. Specifically, the selected location in the image indicates where the robot's forward-facing end effector should be located at after the action has been completed. The agent then uses a movement primitive to execute the move.

We experiment with two types of movement primitives: one that moves in a straight line towards the selected location, and one that follows the shortest path to the selected location. The straight line primitive simply turns the robot to face the selected location, and then moves forward until it has reached the location. The shortest path primitive (well-suited for environments with obstacles) uses the global occupancy map introduced in Sec. III-B to compute and follow the shortest path to the desired target location (Fig. 5). For both, it is possible for the robot to collide with previously unseen obstacles, in which case a penalty (see Sec. III-D) is incurred by the agent. Our experiments in Sec. IV-A compare this representation to discrete action alternatives (*e.g.*, steering commands) commonly used in the literature. Note that in empty environments without obstacles, these two movement primitives are equivalent.

D. Network Architecture and Training Details

We model our parameterized Q-function Q_θ with a fully convolutional neural network (FCN), using a ResNet-18 [13]
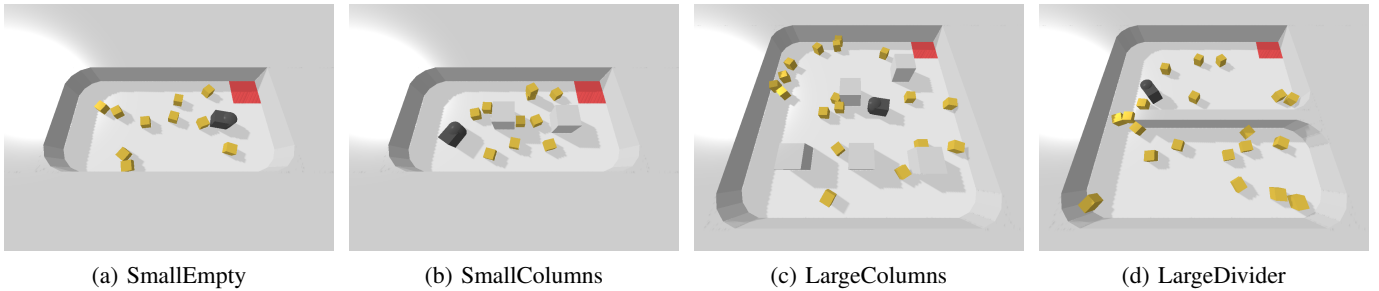
Fig. 6. **Environments.** We tested in four types of environments, each with a receptacle in the top right corner (red), randomized arrangements of objects (yellow blocks), and randomized placements of obstacles (gray) of increasing difficulty (left to right).

backbone for all experiments. The FCN takes as input the 4-channel image state representation (described in Sec. III-B) and outputs a state-action value map (described in Sec. III-C). We removed the AvgPool and fully connected layers at the end of the ResNet-18, and replaced them with 3 convolutional layers interleaved with bilinear upsampling layers. The added convolutional layers use 1x1 kernels, and the upsampling layers use a scale factor of 2. This gives us an output that is the same size as the input. We also removed all BatchNorm layers from our networks, which provided more training stability with small batch sizes. To ensure that the FCN has an adequate receptive field, we designed our observation crop size (96 by 96) such that the receptive field of each network output covers over a quarter of the input image area, and thus always covers the center of the image in which the robot is located.

Rewards. Our reward structure for reinforcement learning (computed after each transition) consists of three components: (1) a positive reward of +1 for each object that ends up in the receptacle (objects in the receptacle are removed thereafter), (2) partial rewards and penalties based on whether each object was moved closer to or further away from the receptacle (proportional to the signed change in distance computed using either Euclidean distances or shortest path distances – comparison in Sec. IV-A), and (3) small penalties of -0.25 for undesirable behaviors (collisions or nonmovement).

Training details. During training, our agent interacts with the environment and stores data from each transition $e_t = (s_t, a_t, r_t, s_{t+1})$ in an experience replay buffer of size 10,000. At each time step, we uniformly at random sample a batch of experiences from the experience replay buffer, and train our neural network with smooth L1 loss (*i.e.*, Huber loss). We pass gradients through only the single pixel in the input state that corresponds to the selected action for a transition [45, 43]. We clip gradients such that they have magnitude at most 10. We train with batch size 32 and use stochastic gradient descent (SGD) with learning rate 0.01, momentum 0.9, and weight decay 0.0001. To account for varying distances traveled for different steps, we apply a discount factor $\lambda = 0.99^{0.25 \cdot \text{dist}}$ with an exponent that is proportional to the distance traveled during that step.

In our experiments, we train for 60,000 transitions, which typically corresponds to several hundred episodes. Each episode runs until either all objects in the environment have been pushed into the target receptacle, or the robot has not pushed any objects into the receptacle for 100 steps. Our policies are trained from scratch, without any image-based pretraining [40]. The target network is updated every 1,000 steps. Training takes about 9 hours on a single NVIDIA Titan Xp GPU.

Exploration. Before training the network, we run a random policy for 1,000 steps to fill the replay buffer with some initial experience. Our exploration strategy is ϵ-greedy with initial $\epsilon = 1$, annealed over training to 0.01 after 6,000 transitions.

IV. EXPERIMENTS

To test the proposed ideas, we run a series of experiments in both simulation and real-world environments. We first describe the simulation experiments, which are used to investigate trade-offs of different algorithms, and then we test our best algorithm on the physical robot.

Task. In every experiment, the robot is initialized with a random pose within a 3D environment enclosed by walls. Within the environment, there is a set of cubic objects scattered randomly throughout free space and a 15 cm by 15 cm square receptacle in the top right corner, which serves as the target destination for the objects. The robot's task is to execute a sequence of actions that pushes all of the objects into the receptacle. Objects are removed from the environment after they enter the receptacle.

Environments. We ran experiments with four virtual environments of increasing difficulty (Fig. 6). The first (SmallEmpty) is a small rectangular environment (1 m by 0.5 m) containing 10 randomly placed objects. The second (SmallColumns) adds a random number (1 to 3) of square (10 cm by 10 cm) fixed obstacles (like "columns") placed randomly. The third (LargeColumns) is a larger version (1 m by 1 m) with more columns (1 to 8) and more objects (20). The fourth (LargeDivider) replaces the columns with a single large divider that is fixed at a randomly chosen y coordinate and spans 80% of the x dimension – this last environment requires the robot to plan how to get from one side of the divider to the other by going through the narrow open gap, and thus is the most difficult.

Evaluation metrics. We evaluate each model by running the trained agent for 20 episodes in the environment it was trained on. Since the environments are randomly generated every episode, for each model, we set the random seed so that the
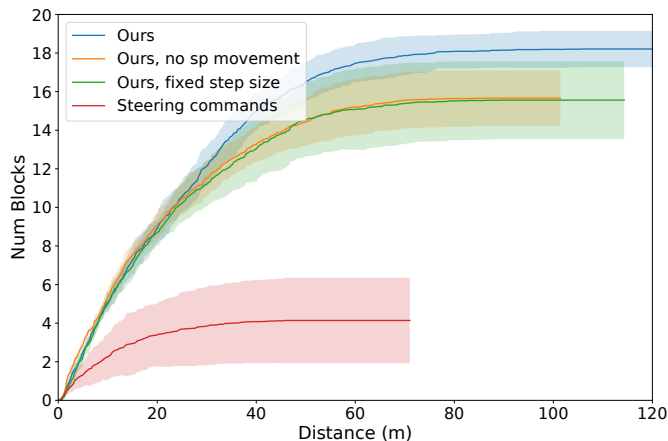
Fig. 7. Number of objects pushed into the receptacle as a function of robot travel distance. These curves show evaluation of experiments that were trained with the LargeDivider environment (shown are means with shaded bars for standard deviations). Our method (blue) is slightly better than its ablation variants with more restricted actions (orange and green), and significantly better than the baseline method trained using steering commands (red).

TABLE I
EFFECT OF ACTION REPRESENTATION

Environment	Ours	No shortest path movement	Fixed step size	Steering commands
SmEmpty	**9.91 ± 0.11**	n/a	9.75 ± 0.20	1.38 ± 0.20
SmColumns	**9.18 ± 0.14**	7.88 ± 0.70	9.05 ± 0.38	0.82 ± 0.33
LgColumns	**18.29 ± 0.45**	14.70 ± 1.52	17.52 ± 0.82	1.20 ± 0.64
LgDivider	**18.23 ± 0.92**	15.66 ± 1.44	15.56 ± 2.01	4.14 ± 2.21

(Number of objects pushed into the receptacle per episode)

exact same set of generated environments are presented to each model, including the initial robot pose, object poses, and obstacle placements. For all experiments, we do 5 training runs with the same setup and report the mean and standard deviation across the 5 runs. Trained agents are evaluated using an ϵ-greedy policy with $\epsilon = 0.01$.

We use two evaluation metrics. The first simply measures the number of objects that have been pushed into the receptacle at the end of an episode (Tab. I). The second plots the number of objects successfully pushed into the receptacle as a function of how far the robot has moved (Fig. 7). Additionally, we compare training sample efficiency by plotting objects per episode on the training environments, as a function of training steps (Fig. 8). Higher numbers are better.

A. Simulation Experiments

Comparison to baseline. Our first set of experiments is designed to evaluate how spatial action maps compare to more traditional action representations. To investigate this question, we ran experiments with an 18-dimensional steering commands action space, with actions representing taking a 25 cm step forward in one of 18 possible turn directions. We created a modified (baseline) version of our system by replacing the last layer of our network with a fully connected layer that outputs the predicted Q-value for each of the 18 actions. We also added two additional channels to the state representation to encode the relative position of every pixel location. This modified
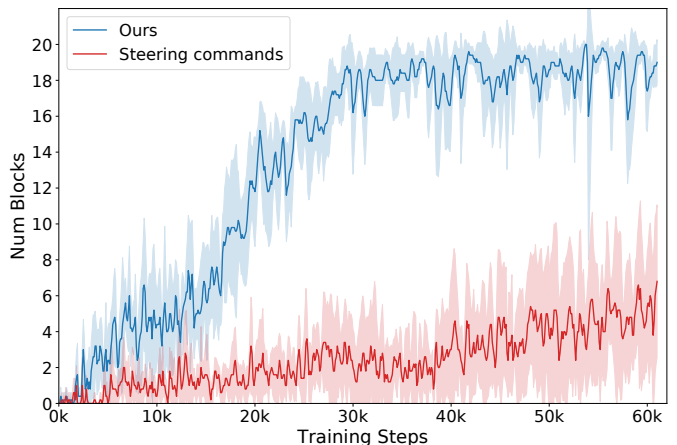

Fig. 8. Training curves for the LargeDivider environment. Our agent trained with spatial action maps (blue) is significantly more sample efficient compared to the baseline that uses steering commands (red).

network mimics the DQN architectures and actions typical of other navigation algorithms [15, 19, 27, 2, 37, 38, 39, 28], and yet is the same as ours in all other aspects.

Results are shown in Tab. I. We find that the steering commands baseline (right) is unable to learn effectively in any of the four environments. We conjecture the reasons are two-fold. First, the baseline network must learn a mapping from observations to discrete actions, which may be harder than the dense prediction enabled by spatial action maps. Second, the baseline agent can only reap rewards by executing a long sequence of short steps, and so it is difficult for the algorithm to achieve any reward at all in the early phases of training (Fig. 8). In contrast, our method uses higher-level actions (that go directly to the selected location), and thus can discover rewards with fewer actions. This results in a policy that performs the task more completely and more efficiently (Fig. 7).

Effect of shortest path movement primitive. To test the hypothesis that using a shortest path movement primitive helps our algorithm learn more efficiently, we ran experiments with a small modification to our system: the robot moves in a straight line to the selected target location (rather than along the shortest path). The results of this experiment ("No shortest path movement" in Tab. I, and orange curve in Fig. 7) show a degradation in performance, particularly when there are more obstacles. Even though the shortest paths are computed with potentially inaccurate occupancy maps, it still seems to be advantageous to use them when navigating around obstacles.

Effect of fixed step size. To test the hypothesis that movement primitives with longer trajectories help our algorithm learn more efficiently, we ran experiments with a different modification to our system: the length of any trajectory is fixed at 25 cm (the same step size as the steering commands baseline). The Q-network makes dense predictions as usual, but at each iteration, the agent steps a fixed distance in the direction of the position with the highest Q-value. The result of this variant ("Fixed step size" in Tab. I and green curve in Fig. 7) shows that taking shorter steps indeed degrades performance. Even

though it would be possible for the agent to take many small steps to achieve the same trajectory as a single long one, we find that it learns more quickly with longer trajectories. We conjecture that this could be due a less direct mapping from visual features to actions, as well as inconsistencies in Q-values predicted from different perspectives, causing the agent to waver between different endpoint targets as it takes many small steps.

Effect of shortest path input channels. In addition to the overhead image, our system gives the agent three additional input image channels: (1) an image with a circle indicating the robot position, (2) an image with shortest path distances from the agent's position, and (3) an image with shortest path distances to the receptacle. To test whether the latter two of these channels are useful, we ran ablation studies for each of them. The results (Tab. II) show that the channels provide little benefit in the environments with smaller obstacles, but help the system train more effectively in the most challenging environment (LargeDivider). We conjecture that providing the shortest path distances can help the agent prioritize target locations with objects that can be reached more easily (without having to go around large obstacles).

Effect of shortest path partial rewards. We train our agents with signed partial rewards given for pushing objects closer to or further away from the receptacle. We hypothesize that in environments with obstacles, it is important to give partial rewards based on changes in shortest path distance rather than Euclidean distance from the receptacle. We verify by running ablations that use Euclidean distance rather than shortest path distance, as shown in Tab. III. We observe that agents trained with Euclidean distance partial rewards indeed perform worse, particularly for the LargeDivider environment, where true shortest path distances to the receptacle can be much larger than Euclidean distances. We believe that giving partial rewards based on shortest path distances provides a better training signal to the agent. While Euclidean distances indicate whether an object is close to the receptacle, shortest path distances additionally factor in the need to push the object around the obstacles to get to the target receptacle. For completeness, we also show the performance when no partial rewards are given.

Effect of removing all shortest path components. Here we remove all shortest path components of our system, specifically (1) shortest path movement primitive, (2) shortest path channels, and (3) shortest path partial rewards. We replace them with their straight-line variants: (1) straight-line movement, (2) channel containing Euclidean distance to receptacle, and (3) Euclidean distance partial rewards. The results are shown in Tab. IV. Indeed we see that for the SmallEmpty environment, there is no significant difference because there are no obstacles present. However, in more difficult environments (LargeDivider), we see that our method is much better at handling the obstacles. The difference can be seen clearly in the example trajectories visualized in Fig. 9.

TABLE II
EFFECT OF SHORTEST PATH INPUT CHANNELS

Environment	Ours	No shortest path from agent	No shortest path to receptacle
SmallEmpty	**9.91 ± 0.11**	9.82 ± 0.21	n/a
SmallColumns	9.18 ± 0.14	9.18 ± 0.32	**9.20 ± 0.22**
LargeColumns	18.29 ± 0.45	18.40 ± 0.88	**18.88 ± 0.49**
LargeDivider	**18.23 ± 0.92**	16.87 ± 1.97	16.71 ± 1.49

TABLE III
EFFECT OF SHORTEST PATH PARTIAL REWARDS

Environment	Ours	No shortest path in partial rewards	No partial rewards
SmallEmpty	**9.91 ± 0.11**	n/a	8.87 ± 1.61
SmallColumns	**9.18 ± 0.14**	9.13 ± 0.28	9.02 ± 0.66
LargeColumns	**18.29 ± 0.45**	17.89 ± 0.97	13.95 ± 3.19
LargeDivider	**18.23 ± 0.92**	16.87 ± 1.97	12.02 ± 1.08

TABLE IV
EFFECT OF REMOVING ALL SHORTEST PATH COMPONENTS

Environment	Ours	Ours no shortest path	Steering	Steering no shortest path
SmEmpty	**9.91 ± 0.11**	9.82 ± 0.10	1.38 ± 0.20	1.23 ± 0.69
SmColumns	**9.18 ± 0.14**	8.05 ± 0.29	0.82 ± 0.33	0.72 ± 0.44
LgColumns	**18.29 ± 0.45**	15.63 ± 1.17	1.20 ± 0.64	0.33 ± 0.20
LgDivider	**18.23 ± 0.92**	10.06 ± 0.89	4.14 ± 2.21	0.26 ± 0.12

Our method pushes objects efficiently along shortest paths trajectories through free space (left). In contrast, the ablative version without shortest path components (middle) is less adept at navigating around obstacles and continually pushes objects up against the divider.

We similarly run the same ablations for the steering commands baseline. Specifically, we remove the shortest path channels and shortest path partial rewards, and replace them with straight-line variants. We find that while the baseline has some ability to handle obstacles when given shortest path channels and shortest path partial rewards, the performance on environments with obstacles can be dramatically worse without these shortest path components (Tab. IV).

B. Real-World Experiments

We conduct experiments on the physical Anki Vector robots by replicating the SmallEmpty simulation environment on a tabletop. Our setup can be seen in Fig. 10. We mount a camera over the tabletop, and affix fiducial markers to the robot and the objects, as well as the corners of the room. Using the overhead camera, we obtain real-time millimeter-level pose estimates of the robots and objects, which we then map into our simulation environment.

In this way, we enable our simulation environment to mirror the real-world environment. This means we can test our policies, which were trained in simulation, directly in the real-world setup. Given a state representation rendered by the simulation, our trained policy outputs a high-level action, which is executed on the physical robot by a low-level controller. Overall, we find that trained agent behavior in the real-world environment is qualitatively similar to the simulation, but not

(a) Ours

(b) Ours w/o shortest path components
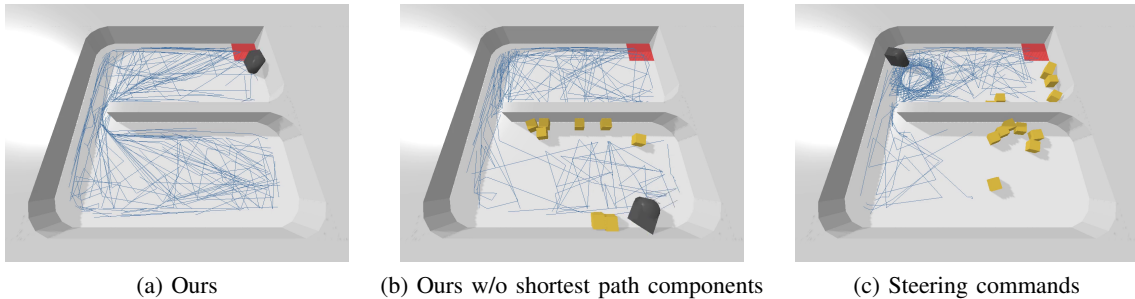
(c) Steering commands

Fig. 9. **Episode trajectories.** In these runs on the LargeDivider environment, our method (a) follows a trajectory (blue line) that clears all objects efficiently. If we disable all shortest path components (b), it pushes objects straight towards the receptacle, and the ones in the bottom half get piled up against the divider. If we use the baseline action representation based on steering commands (c), the agent never learns to navigate effectively in this difficult environment.

quite as efficient due to differences in physical dynamics. We tested our best model on the SmallEmpty environment, and averaged across 5 test episodes, the real robot is able to push 8.4 out of 10 objects into the receptacle within 15 minutes, and all 10 within 30 minutes. We show videos of our robot executing policies (that were learned in simulation) in the real-world environment at https://spatial-action-maps.cs.princeton.edu.

Looking at the videos, it is interesting to observe the emergent behaviors learned by the robot. Perhaps the most common is to first push objects up against the walls, and then later "sweep" multiple objects together along a wall with a single long trajectory ending at the receptacle (note the long paths along the walls in Fig. 9). This behavior is depicted in Fig. 10, where a sequence of actions from one episode is shown in time-lapse as the robot pushes two objects at once (four near the end). Other emergent behaviors include retrying, where the the robot makes multiple attempts to nudge objects towards the receptacle after initial failure.

C. Limitations

Limitations of our setting and experiments. Because of our sim-to-real setup, our results are limited by the accuracy of our simulations (including the measurement accuracy of physical properties in our setup), the quality and robustness of our motion primitive implementations, and our assumptions of perfect localization and mapping. The first two of these are inherent to any system that is trained in simulation, and could be addressed by fine-tuning the learned model in the physical setup. How uncertainty in state and observations should be incorporated into planning is an active research topic, particularly for deep reinforcement learning, and is orthogonal to our investigation into action space representations.

Limitations of spatial action maps. One inherent limitation of spatial action maps is the use of high-level motion primitives, which assume that lower-level control can be handled separately. Concretely, we output a destination that the agent should attempt to reach, and possibly what it should attempt to do once it gets there, and rely on a motion primitive to make that happen. Other action representations, in contrast, may integrate learning of long-range planning and low-level control end-to-end.

Another potential limitation of spatial action maps is that they are only able to represent actions composed of navigating

Fig. 10. **Emergent behaviors.** An Anki Vector robot executes a policy learned in simulation. It first pushes objects towards the wall, then uses the wall as a guide to simultaneously push several objects until they are all in the receptacle.

to a spatial location and performing a discrete action there. While this can be extended to higher dimensional settings (e.g., navigation in 2.5D multi-floor buildings) by using a set of 2D images or 3D grids, there may be tasks which are not compatible with this action representation.

V. CONCLUSION

In this work, we propose "spatial action maps" as a new action representation for mobile manipulation. We study how best to use them in our experimental setting, and show that policies utilizing spatial action maps can be trained much more efficiently than with traditional action representations. We also demonstrate that our agents trained in simulation can transfer directly to a real-world environment without further fine-tuning. This work represents one step in a broader investigation of possible action representations, in one application domain. In future work, it would be interesting to study other related action spaces (e.g., ones that mix navigation and manipulation) and other application domains (e.g., autonomous driving).

REFERENCES

[1] Christopher Amato, George D Konidaris, and Leslie P. Kaelbling. Planning with macro-actions in decentralized pomdps. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014.

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

[3] Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N Siddharth, and Philip HS Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] Tom Bruls, Horia Porav, Lars Kunze, and Paul Newman. The right (angled) perspective: Improving the understanding of road scenes using boosted inverse perspective mapping. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 302–309. IEEE, 2019.

[6] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *International Conference on Learning Representations*, 2019.

[7] Xi Chen, Ali Ghadirzadeh, John Folkesson, Mårten Björkman, and Patric Jensfelt. Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3110–3116. IEEE, 2018.

[8] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[9] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations (ICLR)*, 2017.

[10] Yasuhiro Fuchikawa, Takeshi Nishida, Shuichi Kurogi, Takashi Kondo, Fujio Ohkawa, Toshinori Suehiro, Yasuhiro Watanabe, Yoshinori Kawamura, Masayuki Obata, Hidekazu Miyagawa, et al. Development of a vision system for an outdoor service robot to collect trash on streets. In *Computer Graphics and Imaging*, pages 100–105. Citeseer, 2005.

[11] Wei Gao, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In *Conference on Robot Learning*, 2017.

[12] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *IEEE international conference on computer vision*, pages 2961–2969, 2017.

[15] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474v3*, 2019.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Peter Lehner, Sebastian Brunner, Andreas Dömel, Heinrich Gmeiner, Sebastian Riedel, Bernhard Vodermayer, and Armin Wedler. Mobile manipulation for planetary exploration. In *2018 IEEE Aerospace Conference*, pages 1–11. IEEE, 2018.

[18] Martin Levihn, Jonathan Scholz, and Mike Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Algorithmic Foundations of Robotics X*, pages 19–35. Springer, 2013.

[19] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

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

[21] Douglas Morrison, Peter Corke, and Jürgen Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. In *Robotics: Science and Systems (RSS)*, 2018.

[22] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[23] Takeshi Nishida, Yuji Takemura, Yasuhiro Fuchikawa, Shuichi Kurogi, Shuji Ito, Masayuki Obata, Norio Hiratsuka, Hidekazu Miyagawa, Yasuhiro Watanabe, Fumitaka Koga, et al. Development of outdoor service robots. In *2006 SICE-ICASE International Joint Conference*, pages 2052–2057. IEEE, 2006.

[24] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE International Conference on Robotics and Automation*

(ICRA), pages 1527–1533. IEEE, 2017.

[25] William Qi, Ravi Teja Mullapudi, Saurabh Gupta, and Deva Ramanan. Learning to move with affordance maps. In *International Conference on Learning Representations*, 2020.

[26] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.

[27] Manolis Savva, Angel X Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017.

[28] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *IEEE International Conference on Computer Vision*, pages 9339–9347, 2019.

[29] Pararth Shah, Marek Fiser, Aleksandra Faust, J Chase Kew, and Dilek Hakkani-Tur. Follownet: Robot navigation by following natural language directions with deep reinforcement learning. *arXiv preprint arXiv:1805.06150*, 2018.

[30] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser. Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *arXiv preprint arXiv:1912.04344*, 2020.

[31] Mike Stilman and James J Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics*, 2(04):479–503, 2005.

[32] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *IEEE international conference on robotics and automation*, pages 3327–3332. IEEE, 2007.

[33] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.

[34] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[35] Jur Van Den Berg, Mike Stilman, James Kuffner, Ming Lin, and Dinesh Manocha. Path planning among movable obstacles: a probabilistically complete approach. In *Algorithmic Foundation of Robotics VIII*, pages 599–614. Springer, 2009.

[36] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[37] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018.

[38] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[39] Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357*, 2018.

[40] Lin Yen-Chen, Shuran Zeng, Andy Song, Phillip Isola, and Tsung-Yi Lin. Learning to see before learning to act: Visual pre-training for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[41] Kevin Zakka, Andy Zeng, Johnny Lee, and Shuran Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[42] Brayan S Zapata-Impata, Vikrant Shah, Hanumant Singh, and Robert Platt. Autotrans: an autonomous open world transportation system. *arXiv preprint arXiv:1810.03400*, 2018.

[43] Andy Zeng. *Learning Visual Affordances for Robotic Manipulation*. PhD thesis, Princeton University, 2019.

[44] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[45] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[46] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. In *Robotics: Science and Systems (RSS)*, 2019.

[47] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017.