Adaptive Skill Coordination for Robotic Mobile Manipulation

Naoki Yokoyama[1], Alexander William Clegg[2], Eric Undersander[2], Sehoon Ha[1], Dhruv Batra[1,2], Akshara Rai[2]
[1]Georgia Institute of Technology, [2]FAIR, Meta AI

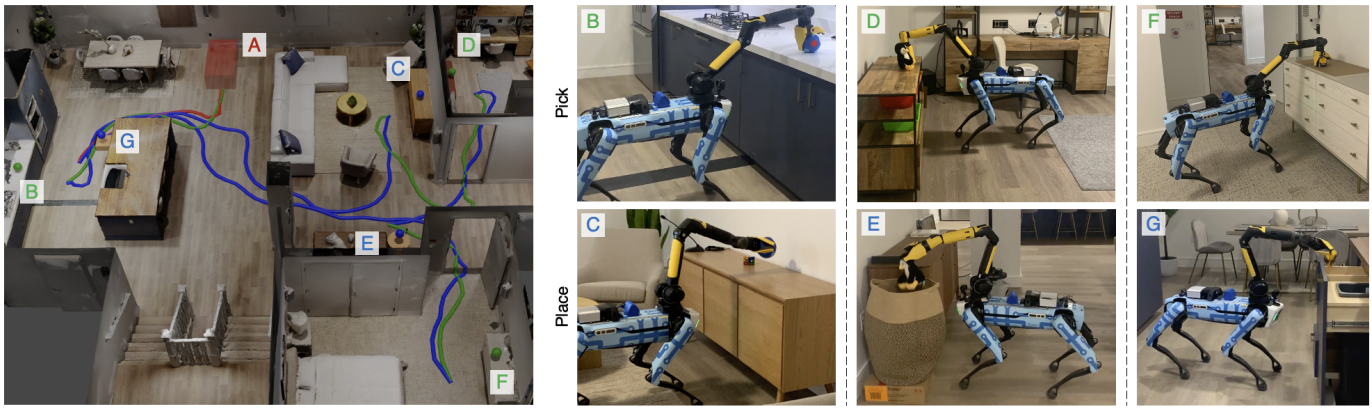


Fig. 1: Adaptive Skill Coordination (ASC) is deployed on Spot in a novel environment and tasked with mobile pick-and-place. ASC operates entirely using onboard sensors – head- and arm-mounted cameras, proprioceptive joint sensors, and egomotion sensors – without access to pre-built maps, 3D models of objects, or precise object locations. Here, Spot navigates from room to room, picking and placing objects, using learned sensor-to-action skills. The robot starts at its dock (red, A), navigates to a pick receptacle (green, B, D, F), searches for and picks an object, navigates to a place receptacle (blue, C, E, G), and places the object at its desired place location, and repeats.

***Abstract*—We present Adaptive Skill Coordination (ASC) – an approach for accomplishing long-horizon tasks (*e.g.*, mobile pick-and-place, consisting of navigating to an object, picking it, navigating to another location, placing it, repeating). ASC consists of three components – (1) a library of basic visuomotor skills (navigation, pick, place), (2) a skill coordination policy that chooses which skills are appropriate to use when, and (3) a corrective policy that adapts pre-trained skills when out-of-distribution states are perceived. All components of ASC rely only on onboard visual and proprioceptive sensing, without access to privileged information like pre-built maps or precise object locations, easing real-world deployment. We train ASC in simulated indoor environments, and deploy it zero-shot in two novel real-world environments on the Boston Dynamics Spot robot. ASC achieves near-perfect performance at mobile pick-and-place, succeeding in 59/60 (98%) episodes, while sequentially executing skills succeeds in only 44/60 (73%) episodes. It is robust to hand-off errors, changes in the environment layout, dynamic obstacles (*e.g.* people), and unexpected disturbances, making it an ideal framework for complex, long-horizon tasks. Supplementary videos available at adaptiveskillcoordination.github.io.**

I. Introduction

A long-envisioned dream of science-fiction literature and foundational Artificial Intelligence (AI) and robotics works is a robot that can assist in everyday tasks like tidying a house, unpacking groceries, cooking, or moving furniture [1]–[3]. While the exact morphology, role, and abilities of such robots differ, they have some key aspects in common – they can perform a large range of tasks, adapt to new and changing environments, and are robust to unexpected, real-world disturbances. In this work, we take a step towards building such general robot assistants that can be deployed in a new, unseen home to solve everyday tasks. Specifically, we study mobile pick-and-place, a subset of general object rearrangement [4]. A robot is initialized in an unseen home and tasked with moving objects from initial to desired locations, emulating the task of 'tidying-up' (Figure 1). The robot operates entirely using onboard sensors – head- and arm-mounted cameras, proprioceptive joint sensors, and egomotion sensors – without privileged information such as pre-built maps, 3D models of objects, or precise object locations. The robot navigates to a receptacle with objects, like the kitchen counter (whose approximate location is known), searches for and picks an object, navigates to its desired place receptacle, and places it.

The long-horizon nature of this task makes learning from trial-and-error challenging: (1) Errors made earlier in the task can prevent the robot from adequately exploring states relevant to later parts of the task. For example, if the robot takes a wrong turn or gets stuck during navigation, it cannot gather experience for picking the target object. (2) Once the robot has learned to complete the initial stages of the task, it may have difficulty learning different skills needed for later stages of the task. For example, if the robot has learned to navigate away from obstacles, it may have difficulty navigating close to a table to pick an object. (3) Different parts of the task might have conflicting goals, requiring careful tuning of reward terms depending on the current stage of the task (*e.g.*, whether the robot is navigating, picking or placing).

To tackle this problem, we present Adaptive Skill Coordination (ASC), which consists of three components: (1) a library of basic visuomotor skills (navigation, pick, and place, shown

in Figure 3a) that take high-dimensional sensory information as input, (2) a skill coordination policy that chooses which skills are appropriate to use when, and (3) a corrective policy that adapts the pre-trained skills when out-of-distribution states are perceived (Figure 3b). Both the coordination and corrective policies take high-dimensional observations as input and are trained using RL and a sparse simple-to-tune reward function (*e.g.*, did the robot succeed at the task?). All components of ASC are trained in diverse, simulated indoor homes, and once trained, it transfers to the real-world 'zero-shot', *i.e.*, without any real-world training data.

We demonstrate the effectiveness of ASC at the long-horizon task of mobile pick-and-place in two unseen real-world environments by deploying it on the Boston Dynamics (BD) Spot robot [5]. Spot is tasked with picking an object from one distant location and placing it at another in a fully-furnished $185m^2$ apartment, and a $65m^2$ university lab. ASC achieves near-perfect performance, succeeding on 59/60 (98%) episodes, overcoming hardware instabilities, picking failures, and adversarial disturbances like moving obstacles or blocked paths. In comparison, sequentially executing the learned skills only succeeds in 44/60 (73%) episodes, due to hand-off errors and inability to recover from disturbances like hardware instability. Our main contribution is a system that can perform real-world, long-horizon tasks in novel environments, without requiring any real-world data. We study its robustness against various adversarial perturbations, such as changing the layout of the environment, walking in front of the robot to repeatedly block its path, or moving target objects mid-episode. Despite being trained entirely in simulation, ASC is robust to such disturbances, making it well-suited for many long-horizon problems in robotics and reinforcement learning.

II. Related Works

Task and motion planning (TAMP). Robotics has a long history of works that study mobile manipulation on humanoid robots [6], [7], wheeled robots [8], and quadrupedal robots [9]. Classical Task and Motion Planning (TAMP) [10] addresses mobile manipulation by forming a task plan, given a *planning domain*, and executing a sequence of model-based skills. However, it assumes access to privileged information like pre-built maps [8], [11], precise object locations [12], [13], and human pilots [6], [7]. Our work uses high-dimensional image inputs, and does not need maps, precise object locations, or human intervention.

Modular learning. Previous works such as [14]–[17] take a modular approach towards learning mobile manipulation. All first learn manipulation and navigation skills, and then execute them sequentially, or use a learned approach to select which to execute. [14] and [15] use a classical STRIPS task planner [18]; both report a substantial amount of failure cases due to hand-off errors (<60% success at picking and placing 5 objects in a row). [17] and [16] train a coordination policy that chooses which skill to execute at the current time step, but [17] does not scale to more complicated indoor environments (0% success in a simulated indoor home as reported by [16]), and [16] assumes access to motion planners that require pre-built maps. Additionally, these works demonstrate results only in simulation, assuming access to privileged information like maps or precise object locations, which are difficult to obtain in the real world. [19] train picking and navigation policies to pick trash (paper balls) in a single room up to $10m^2$ in size. In comparison, we learn to tidy a $185m^2$ fully-furnished unseen real-world apartment, requiring navigation between distant rooms, without a map. [11] study large language models as task planners for mobile manipulation, with learned skills executed sequentially. Our experiments show that such sequential execution suffers from hand-off errors in the real-world. Instead, we train skill coordination and correction policies, which can deal with real-world disturbances and avoid hand-off errors. Moreover, our approach requires no real-world experience, which can be expensive or dangerous to gather (via human teleoperation or policy rollouts).

End-to-end learning. Alternatively, [12], [20]–[23] learn mobile manipulation end-to-end, without leveraging a modular structure. However, these works operate under a strong set of assumptions that substantially simplify the task. [12] constrain their task to a single-room simulated kitchen environment, and assume that the robot has access to the precise location of the object. [20], [22], [23] demonstrate real world results, but simplify the scene; [20] only consider a single corridor with a few obstacles, while [22] and [23] do not consider obstacles in the environment. In contrast, we demonstrate our approach in a real, unseen, fully-furnished apartment without privileged information like maps, precise object locations, or 3D models.

Hierarchical RL and options. Related to TAMP and modular learning is the options framework [24], promising in long-horizon tasks. [25] propose an option-critic that learns to decompose the task into low-level skills and their termination conditions. [26], [27] first train low-level skills, and then train high-level policies that combine them, while [28], [29] train both levels of the hierarchy together, along with a task-specific representation. Broadly speaking, our work is distinguished from this line of work by a combination of real-world mobile manipulation, visual inputs, and lack of access to privileged information. We do not learn task decomposition, which is an open research problem and trivial in our specific setting.

III. Task: Mobile pick-and-place

We consider a mobile pick-and-place task, which we define as finding target objects and moving them to desired place locations. Precise object locations are not known, but each object and its desired place location lie on one of M known receptacles. We follow an episodic coordinate system, established by the robot start location and pose. At the start of each episode, the robot navigates to the closest receptacle with objects, approximately located at $(x, y, \theta)_{pick}$, where x and y are longitudinal and lateral displacements, and θ is the yaw the robot should assume when it reaches the receptacle (see Figure 2). Once the robot reaches the receptacle, it must search for and pick a target object (from a closed-set of object categories, 7 in our setting, shown in Fig. 10). Next, it navigates to

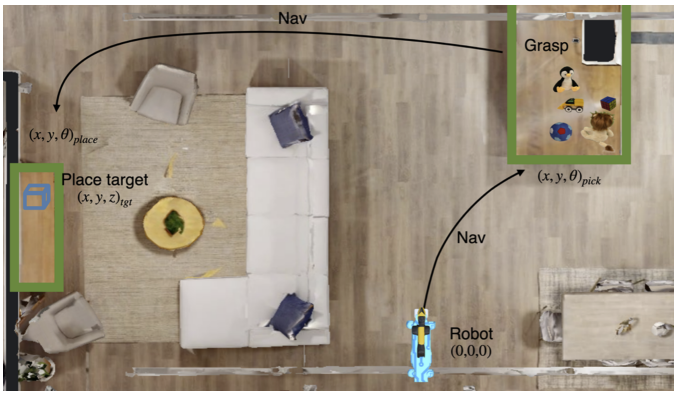


Fig. 2: From its initial location, the robot navigates to a receptacle located at $(x, y, \theta)_{pick}$, searches for, and picks a target object amongst the clutter. It then navigates to the place receptacle corresponding to the object, located at $(x, y, \theta)_{place}$, and places the object at the target place location $(x, y, z)_{tgt}$.

the desired place receptacle $(x, y, \theta)_{place}$ corresponding to the object's category (*e.g.*, all toys go in the hamper), and places the object at the desired place location $(x, y, z)_{tgt}$.

Robot observations. The full observation space of the robot is as follows (illustrated in Fig. 3 and Appendix Fig. 7): (1) two depth images from the front of the robot, concatenated to make a single image I_{front}; (2) a depth image from the gripper of the robot I_{grip}; (3) a bounding box image I_{bbox} around the target object, if detected by an object detector operating on the RGB image of the gripper camera; (4) the joint angles of the robot arm q_{arm}; (5) an egomotion sensor estimating the robot's relative heading and displacement from where it started. We also provide approximate coordinates of receptacles where an object can be picked or placed (*e.g.*, kitchen counter, table, hamper). While this provides robot information about some landmarks in the environment (*i.e.*, approximately where objects can be found), it does not require a map of the environment or any knowledge of obstacle locations, as needed by prior work like [11]. Additionally, exact locations of objects are not needed, unlike prior work such as [14], which is more suitable for real-world experimentation.

IV. Adaptive Skill Coordination

Training ASC consists of two steps. First, we train a library of 3 basic visuomotor skills: navigation, picking, and placing (Figure 3a). Next, we train a skill coordination policy that chooses which skills are appropriate to use when, and a corrective policy that adapts the pre-trained skills when out-of-distribution states are perceived (Figure 3b).

A. Basic visuomotor skills

The navigation, pick, and place skills are trained in diverse simulations of indoor environments using RL. They utilize the Boston Dynamics (BD) API for executing actions in the real world (details in Appendix VIII-E), while significantly improving Spot's capabilities over the BD API (Sec. VI-D).

Navigation skill. To learn the navigation skill, Spot is initialized in a random location in a scene from the HM3D dataset [30] (Figure 4, top), and tasked with reaching within 0.3 m and 5° of a distant goal pose (x, y, θ), specified relative to its start pose (similar formulation to PointGoal navigation [31]). The navigation skill uses concatenated depth images from the robot's front cameras, $I_{front} \in R^{240\times212}$, as well as an egomotion sensor, which provides the robot's current pose relative to its start pose. These observations are a subset of the total observations o received by the robot, and do not include images from the gripper, which may be blocked when holding an object. The output of the navigation skill π_{nav} is a 2-dimensional desired longitudinal and angular base velocity (v, ω) in the robot's local frame:

$$v, \omega \sim \pi_{\text{nav}}(I_{front}, (x, y, \theta)) \quad (1)$$

where v and ω range within [-0.5, 0.5] $^m/_s$ and [-30, 30]$^\circ/_s$, respectively. The navigation reward function encourages the agent to reach the goal while minimizing completion time, collision count, and the amount of time spent moving backwards (similar to [32], detailed in Appendix VIII-C).

Pick skill. The pick skill relies on the BD grasp API (Appendix VIII-E) for real-world grasping, which takes a pixel from the gripper camera located on the target object as input. However, the BD grasp API sometimes fails to grasp the target object, or grasps the wrong object, if it is far from the gripper ($\sim$>0.8 m) or occluded by other objects. To increase the likelihood of success, the goal of the pick skill is to search for the target object, bring the gripper close to it, keep it centered in the gripper camera's view, and then call the BD grasp API. To train such a pick skill in simulation, Spot is tasked with reaching towards an object on one of the receptacles in the ReplicaCAD scene (Figure 4, bottom), starting from random arm joint positions. Multiple objects are initialized on the receptacle, from which one is chosen at random to be the target object, simulating clutter and occlusion. The pick skill receives as input the current joint positions of the arm q_{arm}, and two images: (1) the depth image from the gripper camera, $I_{grip} \in R^{240\times228}$; and (2) a binary mask indicating which pixels are within the bounding box of the target object, $I_{bbox} \in R^{240\times228}$. If the target object is not visible, I_{bbox} is all zeros (no bounding box). The output of the pick skill π_{pick} is desired joint angle displacements Δq relative to the current arm joint positions q_{arm} : $\Delta q \sim \pi_{pick}(I_{grip}, I_{bbox}, q_{arm})$.

The pick reward encourages the gripper to be 0.4m away from the target object, and to point directly at it (similar to [14], Appendix VIII-C), so that the arm is well-positioned for real-world picking. BD grasp is called if the object is within 15° of the gripper camera's forward axis and $0.3-0.75$ m away from the gripper for 4 consecutive policy actions (2 seconds). The BD grasp is simulated by kinematically attaching the object to the gripper if it is in view and close to the gripper, avoiding expensive contact-rich grasping simulation.

Note that the pick skill is not given the precise location of the target object (as in [14]), and must search for it using I_{grip} and I_{bbox}, conducive to real-world deployment. Instead, the robot might know the receptacle that the object is on (e.g., kitchen counter), and has to search for the object there. The

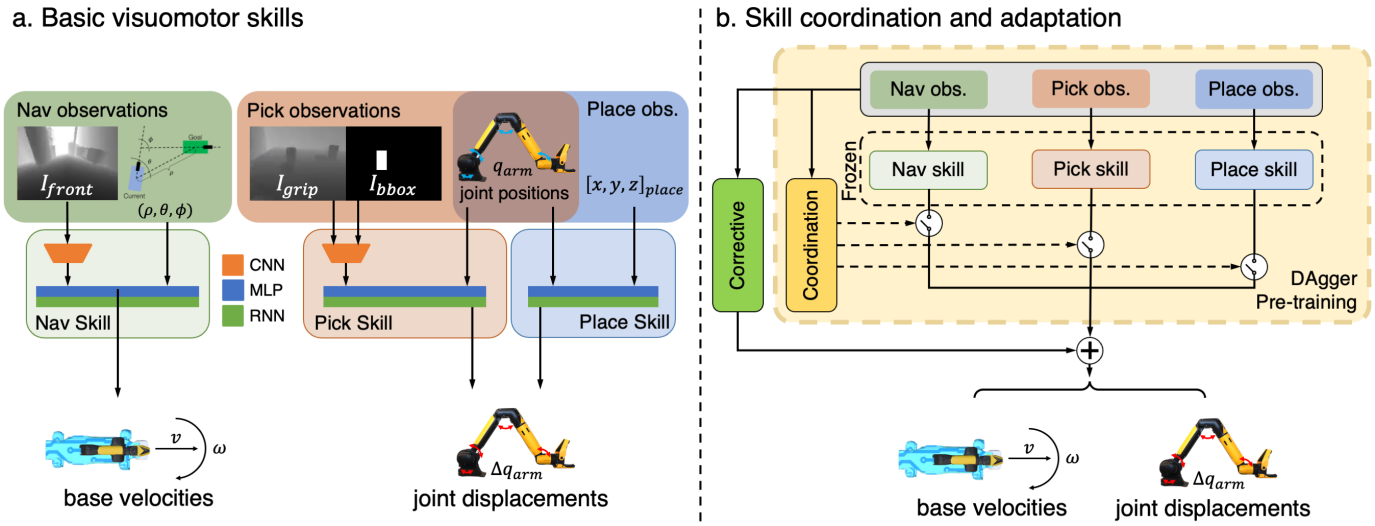


Fig. 3: Training ASC consists of two steps: (a) First, we train a library of 3 basic visuomotor skills in diverse simulated environments. The skills are trained using RL to achieve the relatively shorter-horizon tasks of navigation, picking and placing, and command robot base velocities and delta joint positions. (b) Next, we train a skill coordination policy that chooses which skills are appropriate to use based on observations, and a corrective policy that adapts the pre-trained skills in out-of-distribution states, for the task of object rearrangement.

formulation of I_{bbox} also allows the pick skill to generalize to unseen objects, as long as the bounding box of the object can be detected, such as with Mask R-CNN [33].

Place skill. To train the place skill, Spot is spawned in front of one of 4 receptacles in ReplicaCAD (Figure 4, bottom), while holding an object in its gripper. The robot should place the object within 0.1m of its desired place location $(x, y, z)_{tgt}$, located on the receptacle. Since the gripper camera is blocked by the object and Spot's front cameras are blocked by most receptacles, the place skill does not take visual inputs. The output of the place skill π_{place} are the desired joint angle displacements Δq relative to the current arm joint positions q_{arm} : $\Delta q \sim \pi_{place}(q_{arm}, (x, y, z)_{tgt})$. The place reward encourages the arm to reach the place target with the gripper pointed downwards, so that the object can fall naturally upon opening the gripper (Appendix VIII-C).

In principle, it is possible to replace the place skill with a motion planner that reaches a target end-effector location. However, learning the place skill enables training skill coordination and correction with the place skill in-the-loop.

B. Skill coordination and correction

In the second training phase of ASC, we train a mixture-of-experts coordination policy π^{MoE} that chooses which (pre-trained and frozen) skills are appropriate to use when, depending on observations. However, since the basic skills have never seen the full mobile pick-and-place task, they may have difficulty generalizing to out-of-distribution states observed in the course of this long-horizon task. To handle such cases, we train a corrective policy π^{corr} that adapts the pre-trained skills when out-of-distribution states are perceived.

Skill coordination. Given a library of K pre-trained skills $\Pi = \{\pi_1, \pi_2 \cdots \pi_K\}$, we train a mixture-of-experts coordination policy π^{MoE} which learns to coordinate skills based on observations. Let $g_i \sim \pi^g(\pi_i|o)$ be the probability that skill π_i is selected by a learned gating network π^g given observation o. We consider a binary selection distribution per skill, *i.e.*, $g_i \in \{0, 1\}$, with the skill π_i being selected if $g_i = 1$. We allow multiple or no skills to be selected at the same time, *i.e.*, $0 \leq \sum_i g_i \leq K$. This allows the robot to move its base at the same time as reaching for an object, or operate using only actions from the corrective policy, if appropriate.

Let a_i be the action sampled from skill π_i given the relevant subset of observations o_i. Then $a_{MoE} \sim \pi^{MoE}(o)$ is a mixture of the actions predicted by skills $\pi_{i=1,\ldots,K}$:

$$a_{MoE} = \sum_{i=1}^{K} g_i \cdot a_i, \quad \text{where } g_i \sim \pi^g(\pi_i|o), \; a_i \sim \pi_i(o_i). \quad (2)$$

Note that the action spaces for the basic skills can differ; for example, the pick and place skills control the robot's arm, while the navigation skill controls its base. For skills with complementary action spaces a_{arm} and a_{base}, we create a combined action space $a_{robot} = a_{arm} \oplus a_{base}$, which is a concatenation of the individual skill actions. For skills that share action spaces (like pick and place), we add an additional constraint that only one of the skills can be activated at a time. Specifically, for a subset $a_{sub} \in \{a_{arm}, a_{base}\}$, for all skills π_j whose action space $a_j \in a_{sub}$, we enforce that $\sum_j g_j = 1$. This gives the MoE a hierarchical architecture, as proposed in [34], and separates the regions where skills with shared actions act, while sharing the space between complementary skills. Intuitively, this separates pick and place, while allowing navigation and pick or navigation and place to operate together. In the future, it is easy to extend to several skills that share action spaces, for example multiple place skills for different objects, opening/closing experts, and a learned hierarchical MoE that chooses between them.

Skill correction. Since the skills are trained on simpler tasks, there are likely to be states in the long-horizon task

of mobile pick-and-place where they perform poorly. In such cases, we would like to adapt the skill outputs using a corrective policy π^{corr}, without losing the knowledge gained during the first stage of training. Hence, it is important to only adapt skills in states where they perform poorly. We use the output of the gating network π^g to indicate if a skill should be adapted or not. Intuitively, π^g picks skills that perform well given observation o, *i.e.*, if they were trained on observations similar to o. Conversely, it does not pick skills that perform poorly in the current state, and hence should be adapted. Specifically, we adapt the action sub-space(s) (*i.e.*, base or arm actions) unused by π^g, using corrective actions from the corrective policy π^{corr}. If all skills π_j whose action space $a_j \in a_{sub}$ are not selected, then the adaptive action $a_{adapt} \in a_{sub}$ using $a_{corr} \sim \pi_{corr}(o)$ is:

$$a_{adapt}(o) = \begin{cases} 0, & \text{if } \sum_j g_j \geq 1, \quad g_j \sim \pi^g(\pi_j|o) \\ a_{corr}, & \text{otherwise} \end{cases} \tag{3}$$

The final action a_{ASC}, which is sent to the robot is a sum of the coordination and adaptive actions:

$$a_{ASC} = a_{MoE} + a_{adapt}. \tag{4}$$

a_{ASC} consists of both base and arm actions (v, ω, Δq), and hence can move the base and arm of the robot together. The input to both the coordination and corrective policies is the superset of observations used to train the skills (Figure 3b): $o = \{I_{front}, I_{grip}, I_{bbox}, (x, y, \theta), q_{arm}, (x, y, z)_{tgt}\}$. (x, y, θ) is the location of the desired pick or place receptacle, depending on if the robot has picked an object yet, or not. To process the image observations ($I_{front}, I_{grip}, I_{bbox}$), we reuse the trained vision encoders from the basic skills, and pass the resultant encodings to the coordination and corrective policies. This avoids re-training the visual components, while capturing the necessary visual features as used by the skills. At the same time it gives the coordination and corrective policies information needed for efficient mobile pick-and-place. For example, if the coordination policy sees the target object detected in I_{bbox} before reaching the pick receptacle, it can terminate navigation early and start picking.

V. Training Details

ASC is trained entirely in simulation and deployed zero-shot in two real-world environments. Here we provide the details of the simulations, datasets, reward functions and policy architectures used for training ASC.

Simulators and datasets. We train all our policies in Habitat [14], [35], a fast photo-realistic simulator that supports both navigation and manipulation tasks. The navigation skill is trained on the Habitat-Matterport 3D (HM3D) dataset [30], which contains 1,000 high-quality 3D scans of real-world indoor environments, making it an excellent source of data for training navigation skills (Figure 4, top). However, since the HM3D scans are static, without interactive objects or furniture, they are not suitable for tasks requiring object manipulation. Thus, to train the pick, place, coordination, and corrective

Fig. 4: ASC is trained entirely within simulation. The HM3D dataset is used to train the navigation skill, while ReplicaCAD is used to train the pick, place, coordination, and corrective policies. More visualizations in Appendix Figure 8.

policies, we use the ReplicaCAD scene dataset [14] (Figure 4, bottom). ReplicaCAD features 104 furniture layouts and several designated receptacles (*e.g.*, counters, tables, shelves) on top of which objects can be spawned. A simulated Spot robot is tasked with rearranging an object from the YCB dataset [36] in the ReplicaCAD apartment.

Reward. For the full mobile pick-and-place task, the robot is trained to move one object located on a known receptacle to a target place location on a different receptacle using the ReplicaCAD dataset. During training, the reward is defined as:

$$r_t = w_1 \, \mathbb{I}^{\text{success}} - w_2 \, \mathbb{I}^{\text{collision}} - w_3 \, \mathbb{I}^{\text{backward}} - w_4 \, e_t - C \tag{5}$$

where $\mathbb{I}^{\text{condition}}$ equals 1 if condition is true or 0 otherwise, $success$ indicates whether the target object has been placed at its target location, $collision$ indicates whether a collision occurred, $backward$ indicates whether the robot moved backwards, e_t is the effort exerted in the current step (see Appendix VIII-C), and $C = 0.01$ is a constant penalty accrued at each step to encourage the robot to finish the episode faster. We use weights $[w_1, w_2, w_3, w_4] = [10.0, 0.03, 0.03, 0.0003]$ to balance the different reward components. Appendix VIII-F provides an ablation over reward weights, and in general, we find ASC to be robust to chosen weights.

Learning method. All policies in ASC are trained using deep RL with Decentralized Distributed Proximal Policy Optimization (DDPPO) [37]. Prior works like [14], [38] use dense rewards with extensive reward tuning, while we use a relatively simple reward function with reward terms that are agnostic to subgoals within the task (*i.e.*, navigation/picking/placing goals). However, relying on such a sparse reward function alone can make long-horizon RL challenging due to a large state-action space where most action sequences result in sub-optimal rewards, leading to little to no policy improvement. To overcome this, we utilize a TAMP [10] baseline as a teacher to pre-train the coordination policy using the DAgger algorithm

[39] (see Appendix VIII-G for DAgger details). Next, both the coordination and the corrective policies are trained using RL with the sparse reward from Equation 5, improving ASC's performance over TAMP. Warm-starting the weights of the coordination policy significantly improves ASC's performance over learning from scratch (experiments in Section VI-B).

Policy architecture. Each policy consists of a convolutional neural network (except the place skill, which operates blindly), followed by a multi-layer perceptron and a recurrent GRU [40] layer, and finally a fully-connected layer that parameterizes a diagonal Gaussian action distribution. Specifically, for a policy π that outputs a diagonal Gaussian action distribution given observations o, $a \sim \mathcal{N}(\mu, \sigma\mathbf{I})$, where $[\mu, \sigma] = \pi(o)$. See Appendix VIII-B for more architecture details.

VI. Experimental Evaluation

Next, we present experiments on mobile pick-and-place in the real-world and simulation, comparing ASC against various baselines (Figure 5). ASC is able to coordinate visuomotor skills learned entirely in simulation to achieve mobile pick-and-place in the real-world, without pre-built maps or fine-tuning. It generalizes to unseen environments, and is robust to disturbances such as hardware instability, grasping failures, and adversarial perturbations (*e.g.*, changes in environment layout, dynamic obstacles, moving target objects). We also conduct an ablation analysis over the different components of ASC in simulation, reinforcing their importance.

Real-world environments. We deploy ASC on Spot in two unseen real-world environments: a $185m^2$ fully-furnished apartment (*Apartment*) and a $65m^2$ university lab (*Lab*), shown in Figure 5. No information about these environments was provided during training, and no maps or obstacle locations are provided at test-time. In the *Apartment* the robot needs to take indirect routes to the goal (*e.g.*, navigate out of a room to get to the goal in the next room), while in the *Lab* the robot needs to navigate a cramped space to avoid collisions.

Robot control. In simulation, we use an articulated Spot URDF model provided by BD [5]. We follow the philosophy from [41] for simulating navigation and manipulation. Instead of simulating high-fidelity contact-rich physics, we build an approximate kinematic simulation that learns with the full geometric constraints of the robot, but depends on high-performing low-level BD controllers on hardware. For navigation, the commanded velocities are integrated using a fixed timestep of 0.5 seconds (2 Hz) and the robot is teleported to the resultant waypoint, if there are no collisions with the environment. If the desired integrated pose would cause a collision, the robot is returned to the original pose (no movement occurs) and the policy receives a collision penalty during training. Similarly, the arm is teleported to the next desired arm joint positions if it does not cause collision.

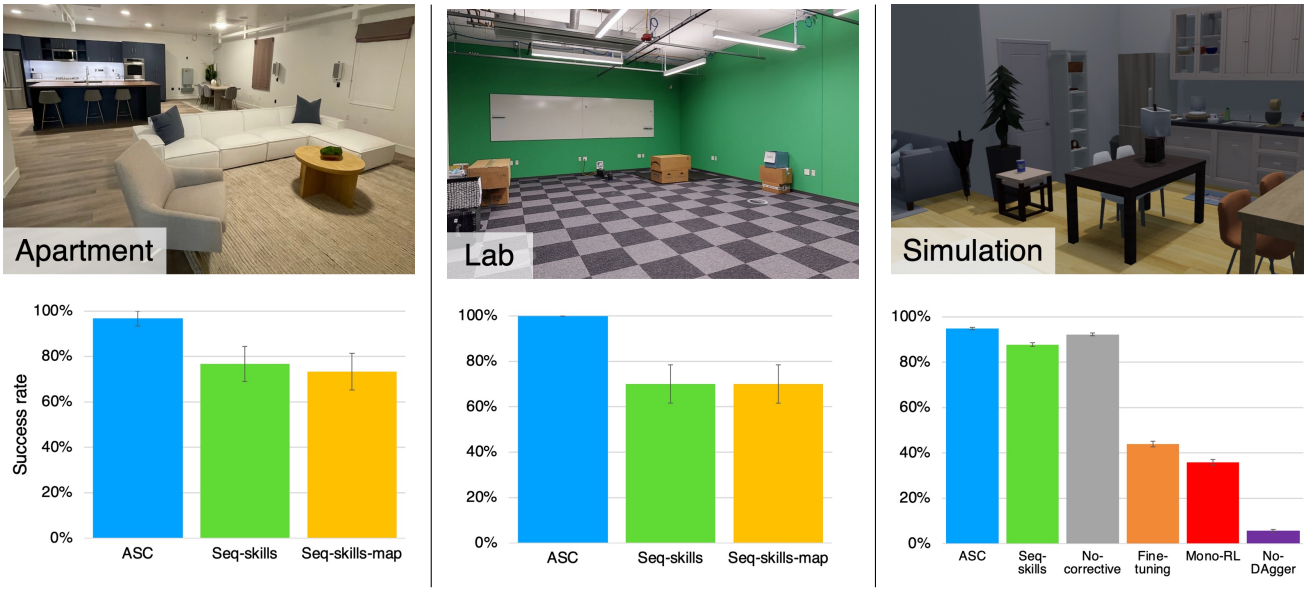
In the real-world, we use the BD API for moving the robot (detailed in Appendix VIII-E). The learned policy outputs actions to the robot at 2 Hz, while observations from the robot are received at a higher frequency of up to 12 Hz. Note that the BD API does not check environment collisions with the arm, and for the base, we turn off obstacle avoidance to allow the robot to move closer to receptacles. Hence, all real-world collision avoidance behavior is achieved entirely through learning in simulation.

A. Mobile pick-and-place in the real world

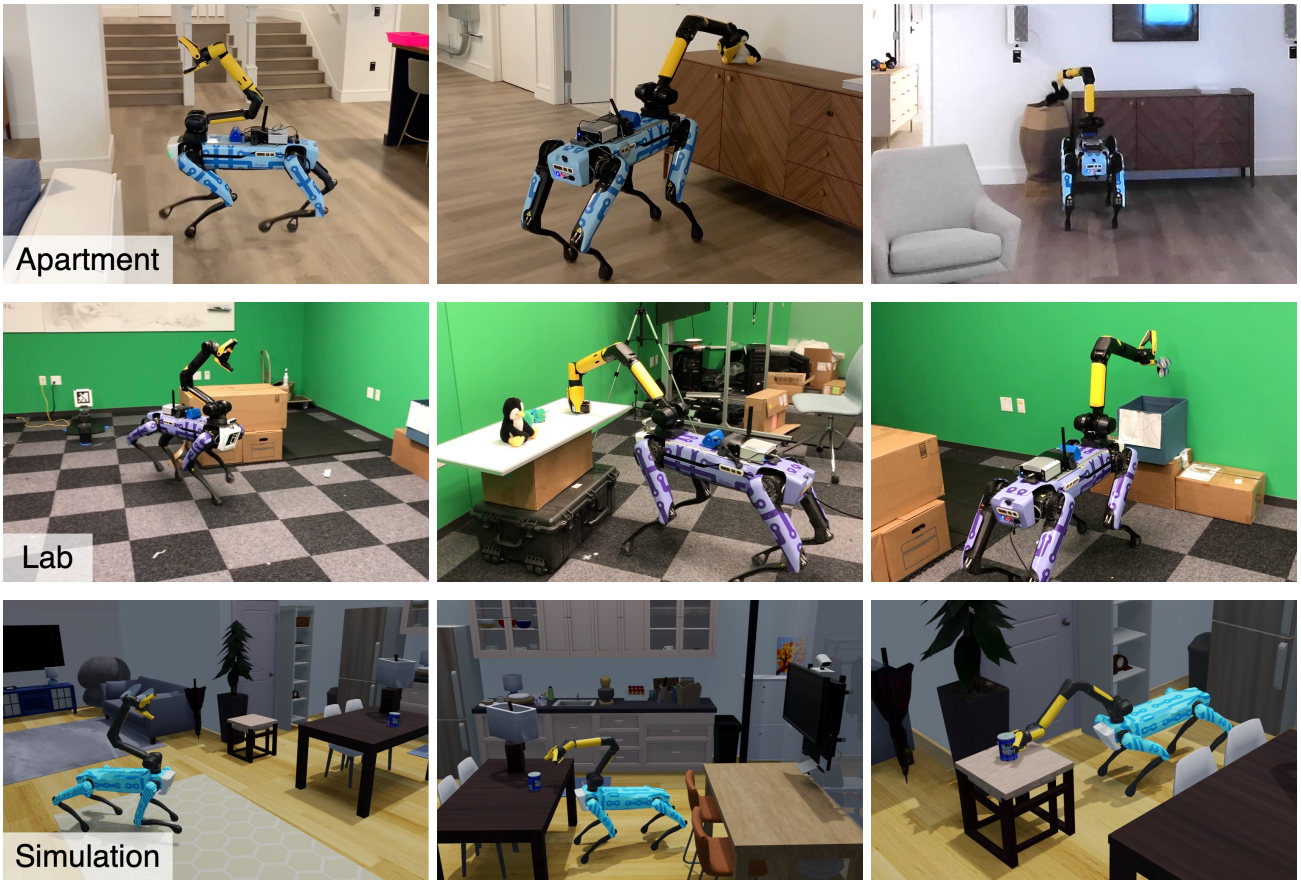
Spot is deployed in the *Apartment* and *Lab* and tasked with rearranging 30 objects in each environment (ranging from a plush penguin/lion/ball, three different toy cars, and a Rubik's cube, shown in Appendix Figure 10). The robot navigates to the closest pick receptacle, searches for and picks an object, navigates to that object's desired place receptacle, and places the object at its desired place location (Figure 1). The process is repeated 30 times, to allow the robot to attempt to rearrange each object once. Figure 1 shows the path of the robot as it rearranges 3 objects in the *Apartment*, moving from room to room while avoiding obstacles. In the real-world, we compare ASC's performance to two baselines:

- Seq-skills: A traditional task and motion planning (TAMP) [10] baseline that creates a task plan, and sequentially executes each learned skill until plan completion. This baseline is sensitive to hand-off errors and real-world disturbances like hardware instability.
- Seq-skills-map: Same as Seq-skills, but the learned navigation skill is replaced with a traditional mapping-based approach. Unlike ASC, Seq-skills-map receives a privileged map of the environment, which is used by a graph-based navigation API provided by BD. This baseline compares traditional mapping-based navigation approaches against learned navigation, and is known to fail if the layout of the environment changes after mapping [42], which learned navigation is robust to.

We measure performance as successful mobile pick-and-place episodes, *i.e.*, the number of objects that the robot successfully picked and placed within 0.1 m of their target place locations over a total of 60 episodes – 30 in *Apartment* and 30 in *Lab*, over 3 runs of 10 episodes each. In *Apartment*, each run of 10 mobile pick-and-place episodes took about 10-15 minutes, and the robot moved approximately 150 m per run. In *Lab*, each run took between 8-10 minutes, and the robot moved approximately 135 m. The robot was completely autonomous throughout all experiments, needing no human intervention. Figure 5a shows a quantitative comparison between ASC and the baselines in the *Apartment* and *Lab*. ASC successfully rearranged $29/30$ objects in *Apartment*, and $30/30$ objects in *Lab*, missing one object due to the BD grasp API failing to pick a toy car. In comparison, Seq-skills successfully rearranged $23/30$ objects in *Apartment* and $21/30$ objects in *Lab*, with the most common failure case being hand-off errors when the navigation skill stopped too far from the receptacle, and the place skill could not reach the desired place location. Seq-skills-map rearranged $22/30$ objects in the *Apartment*, and $21/30$ in the *Lab*, failing due to hand-off errors and once due to a navigation error, where the robot became



(a) Quantitative comparisons in Apartment, Lab, and simulation



(b) Mobile pick-and-place episodes in Apartment, Lab, and simulation

Fig. 5: We evaluate ASC on mobile pick-and-place in two unseen real-world environments and simulation, comparing ASC against various baselines. Our experiments show that ASC is able to coordinate visuomotor skills learned entirely in simulation to achieve mobile pick-and-place in unseen environments, without pre-computed maps or fine-tuning.

stuck and could not navigate to the target location. Seq-skills-map performed similar to Seq-skills, indicating that hand-off errors were not eliminated by utilizing a map-based navigation method instead of learned navigation, as in Seq-skills.. Both approaches performed worse than ASC, which is robust to hand-off errors, re-activating the navigation skill or using the corrective policy to move the robot if needed.

B. Mobile pick-and-place in simulation

We perform additional comparisons in simulation, against the following baselines and ablations of ASC:

- Seq-skills: Same TAMP baseline as on hardware.
- No-corrective: An ablation of ASC with no corrective policy, studying the impact of out-of-distribution states.
- Fine-tuning: An ablation of ASC that fine-tunes visuomotor skills instead of using a corrective policy, using the same reward as ASC. This ablation studies if fine-tuning could be used instead of a corrective policy to adapt pre-trained experts in out-of-distribution states.
- Mono-RL: A single monolithic neural network policy, commanding both the arm and base actions. Mono-RL is pre-trained using DAgger, same as ASC, and fine-tuned using RL with the same reward as ASC.
- No-DAgger: An ablation of ASC which does not pre-train the coordination policy using DAgger.

Details about baselines, and additional ablations over reward weights and policy architecture in Appendix VIII-F.

Figure 5a shows a quantitative comparison over 1500 mobile pick-and-place episodes in simulation (average success and standard error). Learned skill coordination and correction in ASC outperforms Seq-skills ($94.9\% \pm 0.6$ versus $87.8\% \pm 0.8$), by avoiding hand-off errors in both picking and placing. No-corrective is unable to generalize to states which are out-of-distribution for pre-trained skills, and hence performs slightly worse than ASC ($92.3\% \pm 0.7$ for No-corrective versus $94.9\% \pm 0.6$ with ASC). Fine-tuning results in catastrophic forgetting, and performance deteriorates significantly, even with a small learning rate ($43.9\% \pm 1.2$). Mono-RL also performs significantly worse, despite being pre-trained using DAgger, due to the sparse reward in Equation 5 ($35.8\% \pm 1.2$ using Mono-RL). This is consistent with findings in [14] that monolithic RL needs well-tuned rewards to learn complex tasks. No-DAgger performs the worst of all baselines ($5.6\% \pm 1.0$), showing the importance of pre-training with DAgger. These experiments highlight the importance of each component of ASC for robust performance. Specifically, learned skill coordination outperforms skill sequencing, and the corrective policy adds robustness by adapting actions in out-of-distribution states without catastrophic forgetting.

C. Reaction to perturbations

An assistive robot should be reactive to perturbations such as a human walking in front of it, or an object that it was picking getting moved. Since ASC uses reactive visuomotor skills (and is not reliant on an outdated map), it is robust to such disturbances. When faced with changes in environment layout or dynamic obstacles like people, ASC re-routes the robot to a new collision-free path to the goal (Figure 6a, 6b). Since ASC does not take the object position as input, it is also robust to movements of the target object mid-episode. If the object is moved while the robot is searching, ASC continues searching for the object (Figure 6c). Additionally, sometimes the robot becomes unstable when picking an object, prompting BD controllers to move the robot away from the object to re-gain balance. Such cases are out-of-distribution, because ASC is not trained on such disturbances, but ASC re-activates the navigation skill and moves the robot back towards the object, before picking the target object. On the other hand, Seq-skills and Seq-skills-map are not robust to such disturbances, and fail to pick the target object. Robustness to such disturbances is achieved **without any explicit training for perturbations, emerging naturally from the design choices made in ASC.**

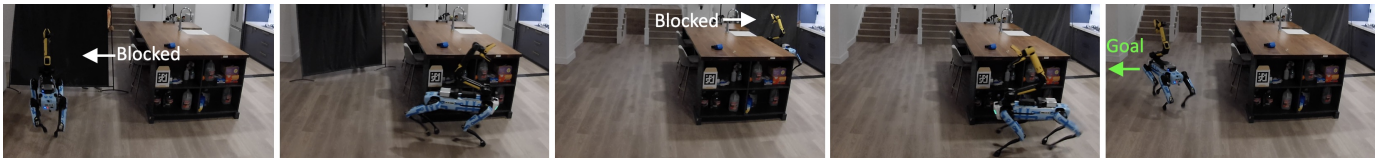
D. Comparing ASC skills and the Boston Dynamics API

	Easy Nav	Medium Nav	Difficult Nav	Picking
ASC skills	100%	100%	100%	100%
BD API	100%	0%	0%	33%

TABLE I: BD API can navigate to unobstructed goals (*Easy Nav*), but fails to circumvent obstacles (*Medium Nav*) or exit a room (*Difficult Nav*). It grasps 33% of objects placed in front of the robot, and cannot grasp occluded objects, or objects not in the camera view. ASC's learned skills significantly enhance the capabilities of Spot, succeeding in 100% of cases. Success is reported over 3 navigation episodes, per difficulty level, and 18 pick episodes. Since the performance of both methods is deterministic, with the same successes and failures across episodes, we don't report standard-error.

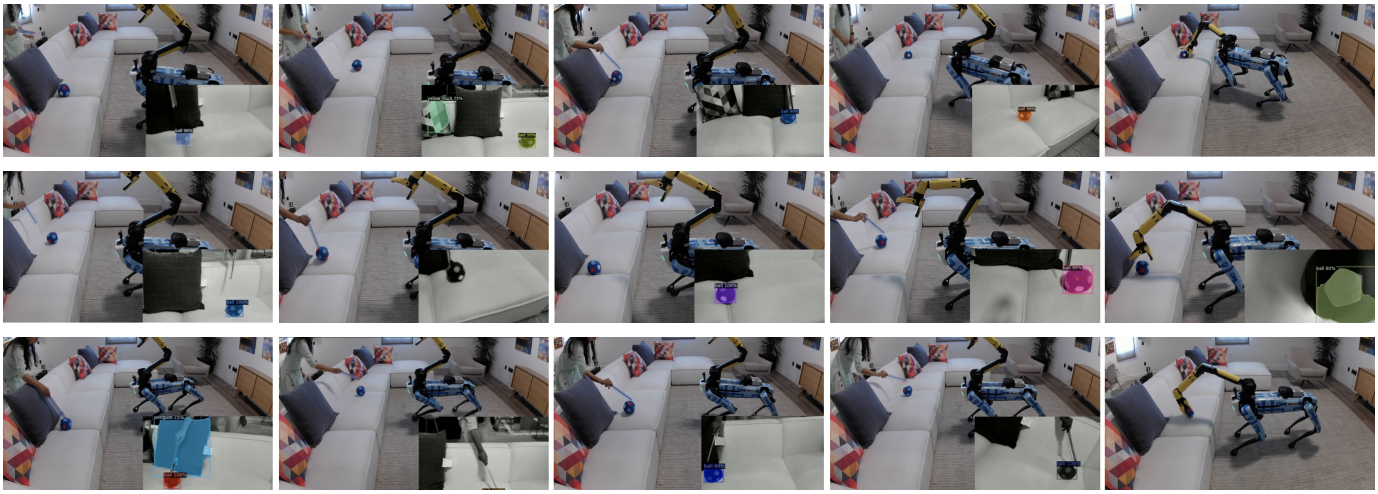
Boston Dynamics (BD) provides navigation, manipulation, and grasping APIs on Spot (Appendix VIII-E), which serve as the backbone for ASC. This has two advantages: (1) ASC is able to leverage controllers specifically designed and tuned for Spot by BD; and (2) it avoids simulating costly and slow high-fidelity, low-level physics, but still enhances robot capabilities through learning in simulation. Here, we show that ASC's learned skills significantly enhance the abilities of Spot, and are essential for consistently succeeding at long-horizon mobile pick-and-place. ASC's learned navigation skill can navigate Spot to distant parts of the environment, while the BD navigation API can only navigate to unobstructed goals when a map is not provided. As shown in Table I, both BD navigation and the learned navigation skill can reach unobstructed goals (*Easy Nav*), but BD navigation fails when asked to circumvent an obstacle blocking its path (*Medium Nav*), or reach goals in another room (*Difficult Nav*). To navigate to these more challenging goals, BD navigation requires a map of the environment, but is sensitive to changes in environment layout [42]. In contrast, the learned navigation skill can perform long-range navigation on Spot in all cases. The learned navigation skill enables navigation in unfamiliar environments, and obviates the need to (1) tele-operate the robot or (2) build a map of the environment.

The BD grasp API typically fails to grasp the target object if it's partially occluded, opting to grasp the occluding object



(a) Robustness to changes in environment layout

(b) Robustness to dynamic obstacles (person blocking the robot's path)



(c) Robustness to moving target object

Fig. 6: ASC is robust to several types of real-world disturbances (best viewed in video at adaptiveskillcoordination.github.io). (a) ASC can reroute the robot upon sudden changes to the environment layout blocking its path, until it successfully finds a path to the goal, even if that path was previously blocked. (b) ASC is also robust to dynamic obstacles, such as moving people in its way, re-routing the robot to the goal. (c) Since ASC does not take object positions as input, it is also robust to perturbations like moving objects. If the object that the robot is targeting is moved mid-episode, ASC continues searching for the object, before localizing and grasping.

instead. In contrast, ASC's learned pick skill searches for objects if they are not initially visible, and moves the arm such that the target object is at the center of the gripper camera's image, before calling the BD grasp. Table I shows that ASC's learned pick skill is able to search for and pick objects in 18/18 episodes, including when occluded, while BD grasp only succeeds in 6/18 episodes, when objects are clearly visible. However, even with these enhancements from ASC, the BD grasp can sometimes fail to pick the target object, due to hardware instability or camera latency (see supplementary video). In such cases, ASC re-activates the pick skill, which resumes searching for the object, and attempts re-grasping. In this way, ASC robustly responds to inadvertent failures of even well-designed low-level controllers.

VII. Conclusion

In this work, we present Adaptive Skill Coordination (ASC) – an approach that coordinates and adapts pre-trained skills based on observations. ASC is deployed zero-shot on the Spot robot at mobile pick-and-place in two unseen, real-world environments, and achieves near-perfect performance, without the need for a map or precise object locations. In comparison, traditional TAMP approaches suffer from hand-off errors, and an inability to recover from disturbances, like hardware instability. Despite being trained entirely in simulation, ASC shows robustness to real-world disturbances, such as dynamic obstacles like people blocking its path, changes to environment layout, and target objects being moved mid-episode. Future work could investigate extensions to a larger number of skills, other mobile manipulation tasks, and high-level reasoning.

REFERENCES

[1] A. I. Center, "Shakey the robot," 1984.

[2] J. H. Connell, "A colony architecture for an artificial creature," MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, Tech. Rep., 1989.

[3] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*. Springer, 2008, vol. 200.

[4] D. Batra, A. X. Chang, S. Chernova, A. J. Davison, J. Deng, V. Koltun, S. Levine, J. Malik, I. Mordatch, R. Mottaghi, M. Savva, and H. Su, "Rearrangement: A challenge for embodied AI," *arXiv preprint arXiv:2011.01975*, 2020.

[5] "Boston dynamics," https://www.bostondynamics.com/spot.

[6] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization based full body control for the atlas robot," in *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 120–127.

[7] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, no. 3, pp. 429–455, 2016.

[8] A. Heins, M. Jakob, and A. P. Schoellig, "Mobile manipulation in unknown environments with differential inverse kinematics control," in *2021 18th Conference on Robots and Vision (CRV)*. IEEE, 2021, pp. 64–71.

[9] Y. Ma, F. Farshidian, T. Miki, J. Lee, and M. Hutter, "Combining learning-based locomotion policy with model-based manipulation for legged mobile manipulators," *CoRR*, vol. abs/2201.03871, 2022. [Online]. Available: https://arxiv.org/abs/2201.03871

[10] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[11] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[12] K. Ehsani, W. Han, A. Herrasti, E. VanderBilt, L. Weihs, E. Kolve, A. Kembhavi, and R. Mottaghi, "Manipulathor: A framework for visual object manipulation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4497–4506.

[13] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5678–5684.

[14] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra, "Habitat 2.0: Training home assistants to rearrange their habitat," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[15] J. Gu, D. S. Chaplot, H. Su, and J. Malik, "Multi-skill mobile manipulation for object rearrangement," *arXiv preprint arXiv:2209.02778*, 2022.

[16] M. Jordan and R. Jacobs, "Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. IEEE, 2021, pp. 4583–4590. [Online]. Available: https://doi.org/10.1109/ICRA48506.2021.9561315

[17] C. Li, F. Xia, R. Martin-Martin, and S. Savarese, "HRL4IN: Hierarchical Reinforcement Learning for Interactive Navigation with Mobile Manipulators," *arXiv e-prints*, p. arXiv:1910.11432, Oct. 2019.

[18] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[19] C. Sun, J. Orbik, C. M. Devin, B. H. Yang, A. Gupta, G. Berseth, and S. Levine, "Fully autonomous real-world reinforcement learning with applications to mobile manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 308–319.

[20] J. Kindle, F. Furrer, T. Novkovic, J. J. Chung, R. Siegwart, and J. Nieto, "Whole-Body Control of a Mobile Manipulator using End-to-End Reinforcement Learning," *arXiv e-prints*, p. arXiv:2003.02637, Feb. 2020.

[21] T. Ni, K. Ehsani, L. Weihs, and J. Salvador, "Towards disturbance-free visual mobile manipulation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 5219–5231.

[22] D. Honerkamp, T. Welschehold, and A. Valada, "Learning kinematic feasibility for mobile manipulation through deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6289–6296, 2021.

[23] C. Wang, Q. Zhang, Q. Tian, S. Li, X. Wang, D. Lane, Y. Petillot, and S. Wang, "Learning mobile manipulation through deep reinforcement learning," *Sensors*, vol. 20, no. 3, p. 939, 2020.

[24] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[25] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[26] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver, "Learning and transfer of modulated locomotor controllers," *arXiv preprint arXiv:1610.05182*, 2016.

[27] T. Li, N. Lambert, R. Calandra, F. Meier, and A. Rai, "Learning generalizable locomotion skills with hierarchical reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 413–419.

[28] O. Nachum, S. Gu, H. Lee, and S. Levine, "Near-optimal representation learning for hierarchical reinforcement learning," *arXiv preprint arXiv:1810.01257*, 2018.

[29] T. Li, R. Calandra, D. Pathak, Y. Tian, F. Meier, and A. Rai, "Planning in learned latent action spaces for generalizable legged locomotion," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2682–2689, 2021.

[30] S. K. Ramakrishnan, A. Gokaslan, E. Wijmans, O. Maksymets, A. Clegg, J. M. Turner, E. Undersander, W. Galuba, A. Westbury, A. X. Chang, M. Savva, Y. Zhao, and D. Batra, "Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. [Online]. Available: https://openreview.net/forum?id=-v4OuqNs5P

[31] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[32] N. Yokoyama, S. Ha, and D. Batra, "Success weighted by completion time: A dynamics-aware evaluation criteria for embodied navigation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1562–1569.

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[34] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[35] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[36] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[37] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames," in *International Conference on Learning Representations (ICLR)*, 2020.

[38] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[39] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[40] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," *arXiv e-prints*, p. arXiv:1409.1259, Sept. 2014.

[41] J. Truong, M. Rudolph, N. Yokoyama, S. Chernova, D. Batra, and A. Rai, "Rethinking sim2real: Lower fidelity simulation leads to higher sim2real transfer in navigation," in *Conference on Robot Learning (CoRL)*, 2022.

[42] "GraphNav Service - Spot 3.2.1 documentation," https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav_service, accessed: 2023-01-29.
[43] J. Ku, A. Harakeh, and S. L. Waslander, "In defense of classical image processing: Fast depth completion on the CPU," in *2018 15th Conference on Computer and Robot Vision (CRV)*, 2018, pp. 16–22.
[44] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.
[45] "Spot SDK, Boston Dynamics," https://dev.bostondynamics.com/, accessed: 2022-09-30.
[46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.
[47] Q. Hou, M.-M. Cheng, X. Hu, A. Borji, Z. Tu, and P. H. S. Torr, "Deeply supervised salient object detection with short connections," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 4, pp. 815–828, 2019.
[48] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.
[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[50] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.
[51] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.
[52] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

VIII. Appendix

The appendix is structured as follows:

- Real-world and simulated environments and observations (VIII-A)
- Network architectures (VIII-B)
- Reward functions (VIII-C)
- Training and hardware evaluation time(VIII-D)
- Boston Dynamics APIs (VIII-E)
- Baselines and ablations (VIII-F)
- DAgger pre-training (VIII-G)
- Object detection pipeline (VIII-H)
- Spot robot hardware (VIII-I)
- Additional robustness experiments (VIII-J)

A. Real-world and simulated environments and observations

Figure 7 provides a visual comparison between the observations ASC uses within simulation and those that it uses when deployed on a real robot. In order to reduce the gap in appearance between the images from real-world depth cameras and those that the robot observed within simulation during training, we denoise the real depth images using the fast depth completion algorithm by Ku et al. [43].

Figure 8 shows the different simulation environments used in the training of ASC. ASC can leverage and fully retain the experience gathered on different datasets simulated within Habitat. The navigation skill is trained on the HM3D dataset [30], while the pick and place skills are trained using the ReplicaCAD [14] and YCB datasets [36]. The HM3D dataset contains 1,000 high-quality 3D scans of real-world indoor environments, making it an excellent source of data for training navigation skills (Figure 8A). We train our navigation skill on 800 of these scans and use the rest as a validation set to evaluate the performance of the navigation expert and select the best checkpoint to use for our navigation skill. However, since the HM3D scans are static, without interactive objects or furniture, they are not suitable for learning pick and place skills. Instead, we use the ReplicaCAD dataset, which has interactive receptacles such as cabinets, drawers, and refrigerators that open/close, for learning the pick, place, coordination, and corrective policies. The robot is tasked with picking, placing or rearranging one of 13 objects from the YCB dataset on one of 4 pieces of furniture (receptacles) in 104 different layouts of the simulated ReplicaCAD apartment. Figure 8B shows 3 different layouts on ReplicaCAD, and Figure 8C shows Spot rearranging an object in each layout.

B. Network architectures

The three basic visuomotor skills and coordination and corrective policies (henceforth collectively referred to as 'policies') have similar network architectures. All policies, except place, start with a visual encoder which takes raw images I as input, and returns an image embedding of size 512 s_I: $s_I = \phi(I)$. The coordination and corrective policies use two visual encoders (one from the trained navigation skill, another from the trained pick skill), whose outputs are concatenated. A visual encoder consists of three 2D convolutional layers, with output channel sizes of {32, 64, 32}, and kernel sizes of {(8,8), (4, 4), (3, 3)}, respectively. The output feature map of these convolutional layers is flattened and passed through a linear layer of output size 512 to obtain the image embedding s_I. Next, s_I is concatenated with the rest of the non-visual observations (depending on the policy), and passed through a 2-layer multi-layer perceptron (MLP) with 512 hidden units (for both layers). The output of the MLP, s_{MLP}, is passed through a gated recurrent unit (GRU) [44] with 1 hidden layer of size 512, and an output layer of size 512. The output of the GRU constitutes the encoded state s_{enc} of the robot, with the recurrent unit accounting for partial observability by storing temporal information. All hidden layers have ReLU activations. The input and output dimension of the GRU network depends on the policy, and is detailed in Table S1.

Policy	Input dim	Output dim
Navigation	512 + 3	2
Pick	512 + 4	4
Place	7	4
Coordination	512 + 512 + 10	3
Corrective	512 + 512 + 10	6

Table S1: Input and output (action) dimensions of each policy.

All policies have four outputs – (1) the current encoded state s_{enc}, (2) the mean μ and (3) standard deviation σ of a diagonal Gaussian distribution, and (4) an estimate of the value of the current state. The encoded state s_{enc} outputted by the GRU layer is passed through three parallel linear output layers (heads); each of these provide the other three outputs (outputs 2-4). The output heads for μ and σ are the size of the action (depends on the policy, listed in Table S1). For a policy π given observations o, an action a is sampled as: $a \sim \mathcal{N}(\mu, \sigma\mathbf{I})$, where μ is the mean, σ is the standard deviation, and $\mathbf{I}$ is the identity matrix. The output layer that predicts the action mean μ is passed through a $tanh$ function, and the variance σ is clipped between 10^{-6} and 1. The third head is a one-dimensional critic head that estimates the value of the current state s_{enc}, used during reinforcement learning to calculate the actor loss for DDPPO [37]. The output layer that estimates the value has a linear activation.

C. Reward functions

Here we detail the rewards used for training basic skills, and the coordination and corrective policies. For the following reward definitions, $\Delta^x_{y,t}$ denotes the geodesic distance (length of the shortest obstacle-free path) between points x and y at time t. $\mathbb{I}^{condition}$ equals 1 if *condition* is true, and 0 if not. The meaning of each condition is listed below:

$\mathbb{I}^{coll}$	whether the agent collided with the environment
$\mathbb{I}^{backward}$	whether the agent moved backwards
$\mathbb{I}^{success}$	whether the agent has successfully finished the task
$\mathbb{I}^{wrong\_pick}$	whether the agent has picked the wrong object
$\mathbb{I}^{\Delta^x_{y,t}<z}$	whether x is less than z meters away from y at time t
$\mathbb{I}^{\Delta^x_{y,t}>z}$	whether x is greater than z meters away from y at time t

Navigation reward. The navigation reward consists of a distance term $dist_t$, which encourages the robot to move closer

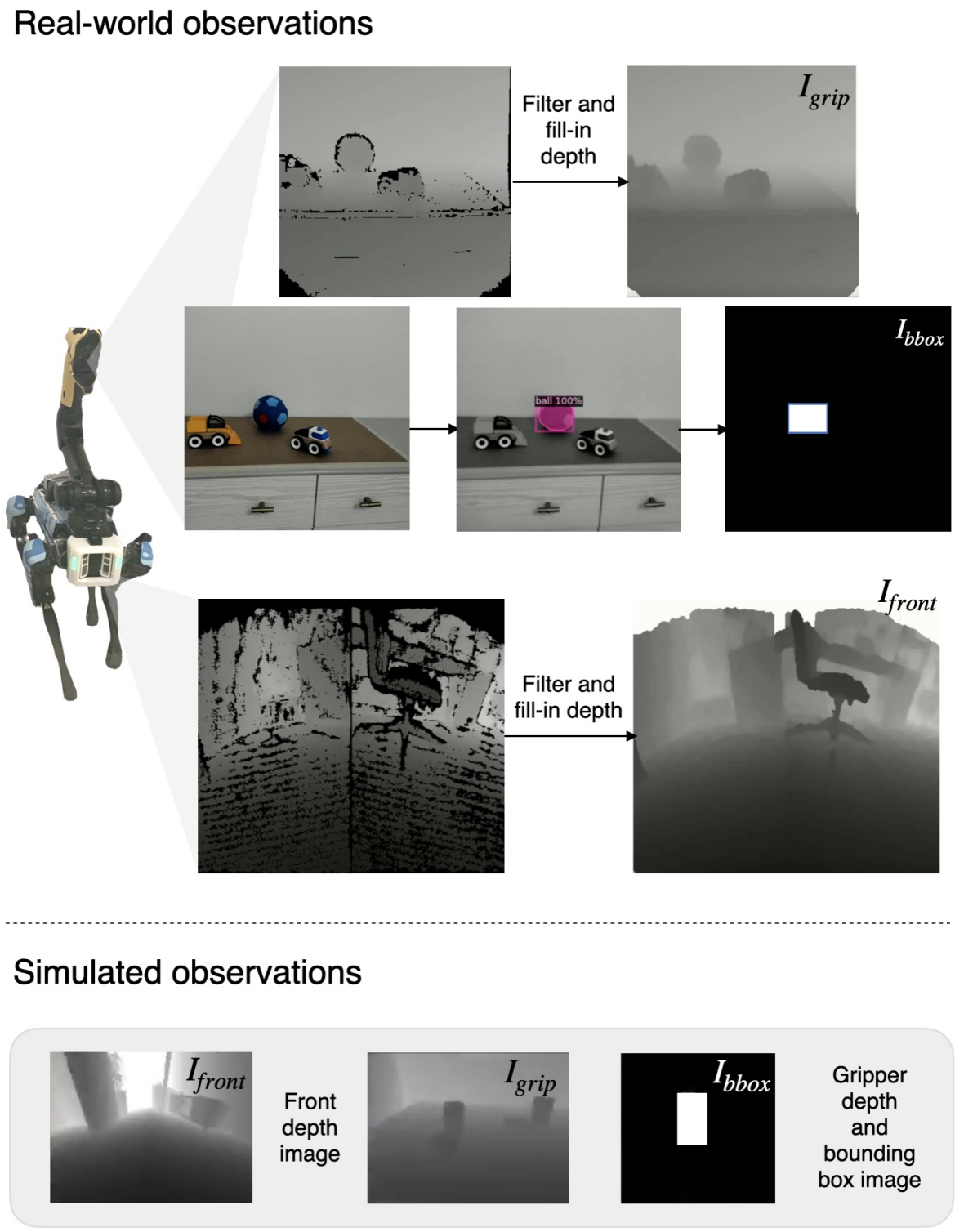


Fig. 7: Simulated and real-world visual observations. The visual observations consist of: (1) two front depth images from the robot, concatenated to make a single image I_{front}; (2) a depth image from the gripper of the robot I_{grip}; (3) a bounding box image around the target object, if detected by an object detector operating on the gripper camera I_{bbox}. To minimize the sim-to-real gap, we de-noise the real depth images using a fast depth completion algorithm that leverages classical image processing [43].

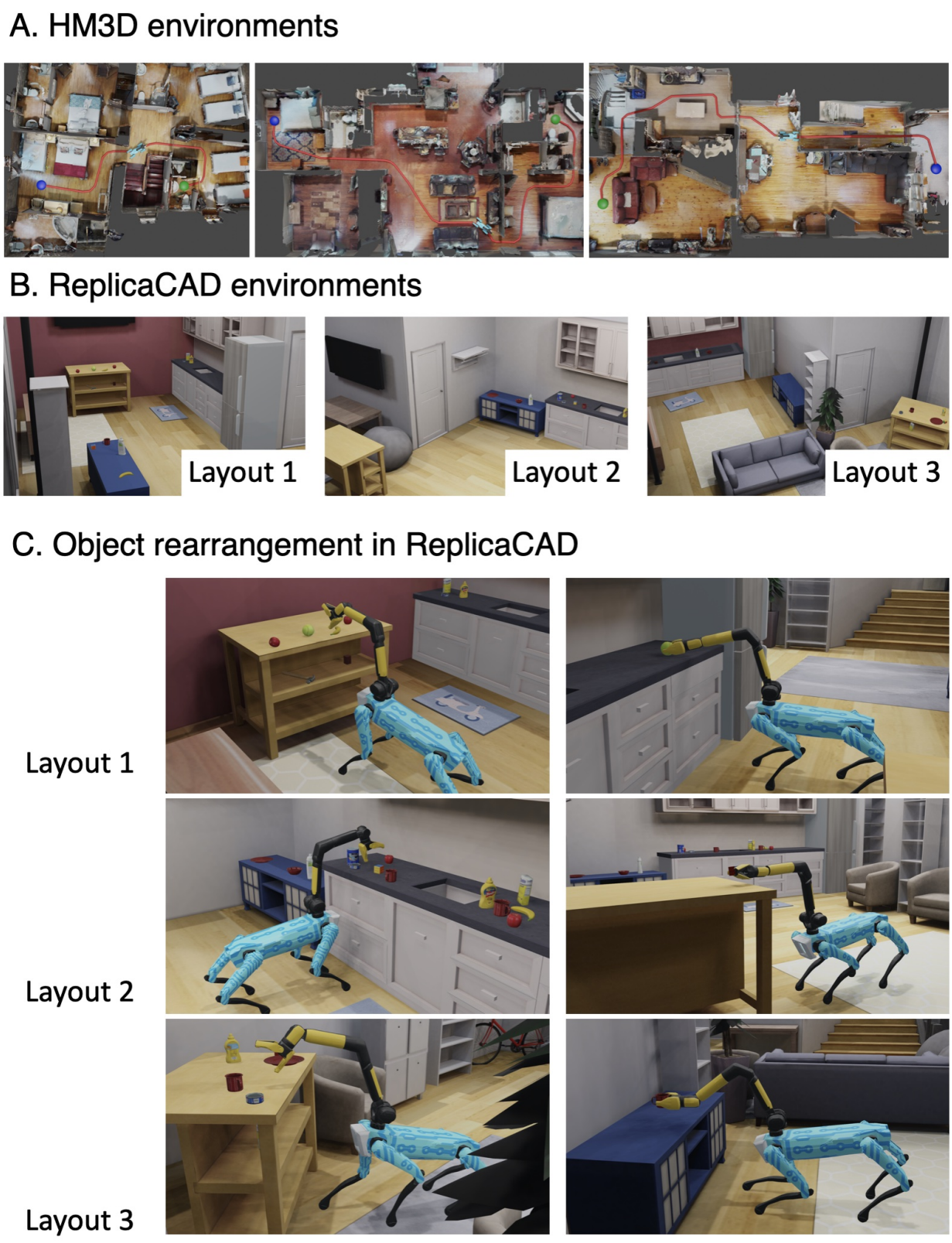


Fig. 8: Simulation environments used during training. The navigation skill is trained on 1000 scans of the HM3D dataset (A), while the pick and place skills are trained in 104 layouts of the ReplicaCAD apartment (B). The coordination and corrective policies are trained on object rearrangement episodes in 104 different layouts of the simulated ReplicaCAD apartment (C).

to the goal, and assume the target orientation θ, by minimizing absolute distance from its current orientation θ:

$$dist_t = \Delta^{goal}_{robot,t} + 0.1\Delta^{\theta^*}_{\theta}\mathbb{I}^{\Delta^{goal}_{robot,t}<1.0} + \pi\mathbb{I}^{\Delta^{goal}_{robot,t}>1.0}$$

When the robot is far from its goal, $dist_t$ does not penalize the robot for deviating away from the target orientation, but instead gives a constant orientation penalty of π radians. Hence the robot can assume any orientation that brings it close to the goal, and is encouraged to reorient itself once it is within 1m of the goal.

Additionally, the navigation reward penalizes collision, backward movement, and gives a terminal reward if the episode was a success:

$$\begin{aligned} r_t = (dist_{t-1} - dist_t) - 0.003\ \mathbb{I}^{coll} \\ - 0.003\ \mathbb{I}^{backward} + 5\ \mathbb{I}^{success} - 0.01 \end{aligned} \tag{6}$$

The 0.01 is a slack penalty which encourages the robot to finish the episode as fast as possible to avoid accruing a negative penalty with each step.

Pick reward. The pick reward uses a distance term, like navigation reward, but instead encouraging the gripper camera to look at the target object, by minimizing the angle between gripper camera's central forward axis and the vector connecting the gripper to the target object at time t, $\theta^{obj}_{grip}(t)$. It also encourages the gripper to keep a distance of about 0.4m from the object:

$$dist_t = \theta^{obj}_{grip}(t) + |\Delta^{obj}_{grip}(t) - 0.4|$$

Additionally, the pick reward penalizes picking the wrong object, gives a successful termination reward, and a slack penalty:

$$r_t = 20\ (dist_{t-1} - dist_t) - 5\ \mathbb{I}^{wrong\_pick} + 10\ \mathbb{I}^{success} - 0.01$$

Place reward. The place reward uses a distance term, which encourages the robot to reach its target place goal. It also encourages the gripper to change its current orientation θ to point downwards before the object is released, so that the object naturally falls on the target location, when the gripper is close to the target (less than 0.4m).

$$dist_t = \Delta^{goal}_{grip,t}$$

$$ori\_dist_t = \begin{cases} \Delta^{-z}_{\theta,t} & \text{if} \quad \Delta^{goal}_{grip,t} < 0.4 \\ \pi & \text{otherwise} \end{cases}$$

where $-z$ signifies the orientation of the negative z-axis in the local robot frame. The place reward penalizes colliding with the environment, gives a successful termination reward, and a slack penalty:

$$\begin{aligned} r_t = 5\ (dist_{t-1} - dist_t) - 0.003\ ori\_dist_t \\ - 0.03\ \mathbb{I}^{coll} + 25\ \mathbb{I}^{success} - 0.01 \end{aligned} \tag{7}$$

Object rearrangement reward. The reward used for training the robot on the long-horizon task of object rearrangement is:

$$r_t = w_1\ \mathbb{I}^{success} - w_2\ \mathbb{I}^{collision} - w_3\ \mathbb{I}^{backward} - w_4\ e_t - C$$

The energy penalty e_t is computed as the sum of the squares of the commanded base velocities and delta joint arm commands. Because the robot is simulated kinematically within simulation during training, we do not use torques. This penalty helps encourage the policy to avoid excessive motions and oscillations in achieving the task. $[w_1, w_2, w_3, w_4, c] = [10, 0.03, 0.03, 0.0003, 0.01]$ in our experiments, and we provide ablation experiments in Section VIII-F. The magnitudes of the weights are calculated to balance the different components in the reward.

D. Training and hardware evaluation time

For training the navigation skill, we utilize 8 GPUs (Nvidia Titan Xp, or similar) with 24 workers each, for a total of 192 workers collecting experience in parallel. We train for 190 million steps for about 14 hours. For the pick and place skills, we utilize 1 GPU with 32 workers and train for 18 million time steps, which takes approximately 22 hours. For training the coordination policy using DAgger, we also use 1 GPU with 32 workers, and train for 1.3 million steps for about 3 hours. Finally, for training both the coordination and corrective policies using deep reinforcement learning, we use 8 GPUs with 8 workers each, and train for 150 million steps for about 45 hours.

E. Boston Dynamics APIs

Spot comes with mature navigation and manipulation APIs provided by Boston Dynamics for controlling the robot. These APIs form the backbone hardware controllers of ASC. Here we describe the APIs used by ASC in detail. For more details, we refer to the BD Spot SDK [45]. Our supplementary video (available at adaptiveskillcoordination.github.io) compares the performance of these APIs against ASC's learned skills, and shows that ASC significantly enhances the capabilities of Spot through learning in simulation.

Navigation API. The navigation API we use to move the robot takes in linear and angular velocity commands for the base of the robot, and moves the the robot to achieve the commanded velocity. If an obstacle is blocking the robot's path, this API stays in-place, and does not move the robot around it. As a result, it can reach unobstructed goals, but has difficulty reaching obstructed and distant goals, for example in another room. However, ASC's learned navigation skill can perform long-range navigation using the velocity tracking API, enhancing the robot's capabilities.

For the Seq-skills-map baseline, we use the NavGraph API provided by BD, which can navigate to distant goals, but requires a pre-built map. This involves manually driving the robot to record various paths throughout the environment in order to build a map, before navigating. The map is built once, and assumed static. As a result, unlike our learned

navigation skill, the NavGraph API is susceptible to changes in the environment; if the environment changes significantly between building a map and testing (for example, if a previously blocked path is opened, or a previously unblocked path is blocked), the map is not updated, causing inefficient paths or even failure. In contrast, our learned navigation skill requires no pre-built map, and can re-route the robot around dynamic obstacles, and is thus robust to such changes in the environment.

The navigation API also provides an egomotion estimate, or the robot's current relative distance and heading from where it started. Retrieving this data does not require any prior knowledge or exploration of the environment; the estimate is computed using onboard odometry (no external cameras or sensors) provided by Boston Dynamics.

Grasp API. The grasp API provided by BD can grasp unknown, arbitrary objects, if provided a pixel lying on the object's image in one of the robot cameras. For ASC, we use the robot's gripper camera image, and send the center of the target bounding box to the grasp API as a pixel lying on the object image. The grasp API then executes whole-body motion planning to grasp the object, and typically succeeds as long as the object is visible, unoccluded, and close to the gripper. These conditions are all met when ASC's pick skill is used to position the arm first. If the BD grasp API fails to grasp an object, it typically returns an error (for example, when manipulation planning fails). In such cases, ASC can re-activate the pick policy and resume searching for objects again.

Manipulation API. The manipulation API is given a target pose of arm joints, as well as a specified amount of time in which to reach this target pose. In our experiments, we give the API 0.5 seconds. At each time step, ASC moves each joint up to 10°, for a maximum angular velocity of 20° per second for each joint. In the future, we could experiment with sending trajectories, or higher frequency velocity commands, both of which are supported by the manipulation API.

F. Baselines and ablations

For the metrics depicted in Figure 3C, we train three policies from three random seeds, and compute mean and standard error for each approach. Policy weights used in evaluation are selected based on their performance on a held-out validation set of 1500 mobile manipulation episodes. All approaches utilize the same superset of skill observations o, and the same action space (v, ω, q_{arm}).

Seq-skills behaves in the following manner: if the robot is not within the success radius and heading (0.3m and 5°) of the navigation goal (x, y, θ), its actions are determined by the navigation skill; otherwise, they are determined by the pick skill if the robot hasn't picked the object yet, or the place skill if it has. The navigation target is reset every time the agent successfully picks or places the object. It is changed either to the location of the target place receptacle once the target object has been picked, or the next clutter receptacle if the object has been placed.

No-corrective is an ablation of ASC in which the corrective policy is not used; it can only select which pre-trained skills to use and execute their actions. It avoids hand-off errors by learning coordination, unlike *Seq-Skills*.

Mono-RL utilizes a policy architecture that consists of two visual encoders (one for I_{front}, another for $I_{grip} + I_{bbox}$) and a downstream MLP-GRU policy with the same architecture described in Appendix VIII-B. Its action distribution outputs both base and arm actions. Like ASC, we utilize DAgger pre-training (see Appendix VIII-G) to warm-start the weights of *Mono-RL* before using deep reinforcement learning with the long-horizon reward function (see Equation 7 in the Materials and Methods section). However, due to the sparsity of the reward, *Mono-RL* is unable to learn the task, despite the DAgger initialization.

We also experiment with removing the recurrent components of the coordination and corrective policies. We see a decrease in performance when the GRU layers are replaced by MLP layers (from 94.9±0.6 to 93.3±0.7, across three seeds). We also evaluate both GRU-based and MLP-based versions of ASC on a harder dataset, in which the navigation goals for both picking and placing locations are set farther away from the corresponding target. In the normal dataset, the picking or placing target can be up to 0.45 m and 30° away from their corresponding navigation goal, whereas they can be up to 1 m and 60° away in the harder dataset. On the harder dataset, we see a larger decrease in performance with using an MLP instead of a GRU (from 65.4±1.2 to 60.7±1.3). These results are summarized in the table below:

Dataset	GRU-based	MLP-based
Normal	$94.87_{\pm 0.57}$	$93.27_{\pm 0.65}$
Hard	$65.4_{\pm 1.23}$	$60.73_{\pm 1.26}$

Lastly, we run an ablation over the reward weights used in Equation 5 and analyse the sensitivity of ASC's performance to these changes. We use 3 sets of weights, and find the performance of ASC remains similar across them.

Weights	Performance
$[10, 0.03, 0.03, 0.0003]$	$94.87_{\pm 0.57}$
$[5.0, 0.01, 0.01, 0.0001]$	$92.8_{\pm 0.67}$
$[15, 0.05, 0.05, 0.0005]$	$93.5_{\pm 0.64}$

G. DAgger pre-training

In DAgger training, a 'student' policy is trained using supervised learning to behave like a 'teacher' policy that provides the target actions at each time step. However, unlike behavior cloning, the actions applied in the environment during training are the student actions, reducing the distribution shift when the student policy is deployed without the teacher. For ASC, we use DAgger to train the gating network π_g using the same logic that drives the *Seq-skills* (see Appendix VIII-F) baseline. π_g is provided with one-hot encoded vectors that identify which skill

it should use at each step. Specifically, if the robot is not within the success condition for navigation, the action label that π_g is trained to output would correspond to activating just the navigation skill; otherwise, if the robot hasn't picked the object yet, the label corresponds to activating just the pick skill, else the label corresponds to activating just the place skill. There are no labels for the corrective policy, hence it is not warm-started using DAgger. Both coordination and corrective policies are then fine-tuned using deep reinforcement learning.

For pre-training the *Mono-RL* baseline using DAgger (before deep reinforcement learning fine-tuning), the labels are the teacher base or arm actions, instead of one-hot encoded vectors. Once the correct skill for the current time step is identified by the *Seq-skills* logic, we use the output of that skill as teacher actions. Because *Seq-skills* only controls either the base or the arm (but not both) at each time step, the part of the action label corresponding to the unused portion of the action space is set to zero.

We find that pre-training with DAgger substantially improves success rate compared to using RL alone for both ASC and *Mono-RL*.

H. Object detection pipeline

To train our object detector, we generate a dataset of automatically annotated images by overlaying object contours on to background images from the COCO dataset [46] (see Figure 10). To extract the object contours, we rely on a U^2-Net [47] model that leverages ReSidual U-blocks (RSU) in its architecture. The U^2-Net is applied to a video of each object, in which the object was seen from a wide variety of viewing angles, which gives us segmented images of the object from different angles. We do not train or fine-tune this model, and instead use a pre-trained set of weights released by the authors. Next, we randomly resize, rotate, flip and superimpose these extracted object contours on 100,000 images from the COCO object detection dataset, to create an automatically labeled dataset. The dataset contains automatically labeled segmented masks of objects, as well as distractor objects originally present in the COCO dataset, and can then be used for training an object segmentation model such as Mask R-CNN [33]. To train our Mask R-CNN model, we use the Detectron2 library of detection algorithms by Meta AI [48]. We use a backbone that leverages ResNet-101 [49] and feature pyramid networks (FPNs) [50]. At the start of training, the network is initialized with pre-trained weights that are available in Detectron2's model zoo. We find that converting all training images to grayscale and Upon deployment for real world experiments, our Mask R-CNN model takes about 200 milliseconds to generate object detections using the gripper's RGB images.

I. Spot robot hardware

ASC is deployed zero-shot to the real-world on a Spot robot [5]. Spot is equipped with five D430 stereo cameras on its base (ASC only uses the two front-facing cameras). In addition, it has a 6 degrees-of-freedom arm and a jaw gripper equipped with an RGB-D camera. The robot comes with mature low-level controller APIs (details in Appendix VIII-E); ASC makes use of Spot's egomotion estimates, its velocity controllers for executing navigation actions, and its grasp API to pick the target object. All ASC policy outputs are sent at 2 Hz to the robot, using a computer equipped with an RTX 3070 GPU, while observations from the robot are updated at about 12Hz asynchronously. This allows the policies to receive the most recent observation as input to reason about robot actions, and hence be reactive to real-world disturbances like moving obstacles, objects, etc.

J. Additional robustness experiments

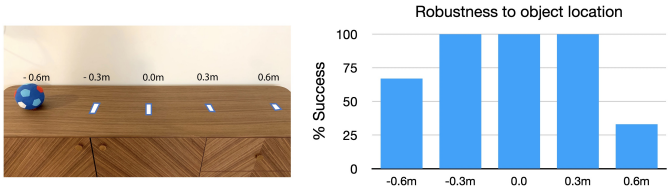


Fig. 9: Robustness to object location

A natural form of disturbance in object rearrangement is the location of the object on a receptacle. For example, the object could be located near the center of the receptacle (e.g., 'hall table'), making it easy to find and grasp it, or it might be at the edge of the receptacle, and the robot has to search for it before grasping. Since the rearrangement task provides only the receptacle location and not the precise location of the object, any approach must be robust to such environmental disturbances. Figure 9 shows an experiment where we move the object further and further away from the receptacle center and measure ASC's performance at successfully grasping and placing the object. We observe that ASC can successfully grasp the object in 9/9 episodes for small perturbations when the object is placed $< \pm 0.3$m away from the receptacle center, and 3/6 times for large perturbations when the object is ± 0.6m away from the center (Figure 3B). Robustness to small perturbations is achieved without any explicit training for perturbations, and emerges from the design choices made in ASC. However, ASC's performance on larger systematic disturbances can be further improved by training ASC with disturbances in simulation, as is common practice in robot learning works like [38], [51], [52].

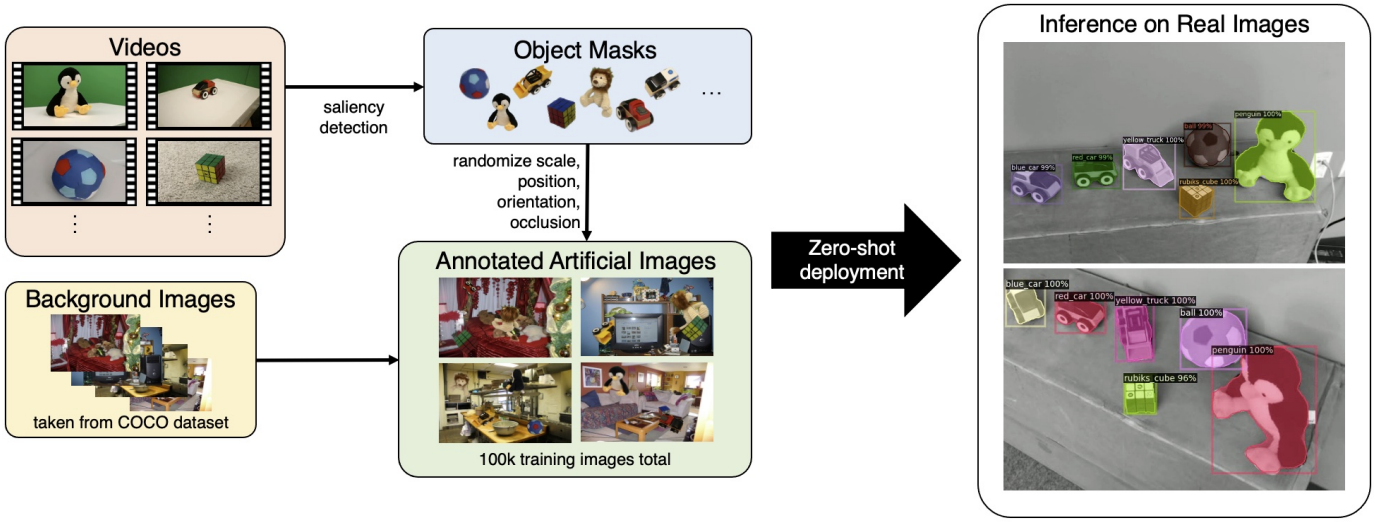


Fig. 10: Training pipeline for the real-world object detector. We generate 100k automatically annotated artificial images for training an object detector using only unlabeled videos of objects and images from the COCO dataset. The detector is then deployed zero-shot to images taken from the real robot's gripper camera.