ASC: Adaptive Skill Coordination for Robotic Mobile Manipulation

Naoki Yokoyama[1], Alex Clegg[2], Joanne Truong[1], Eric Undersander[2], Tsung-Yen Yang[2], Sergio Arnaud[2],
Sehoon Ha[1], Dhruv Batra[1,2], Akshara Rai[2]
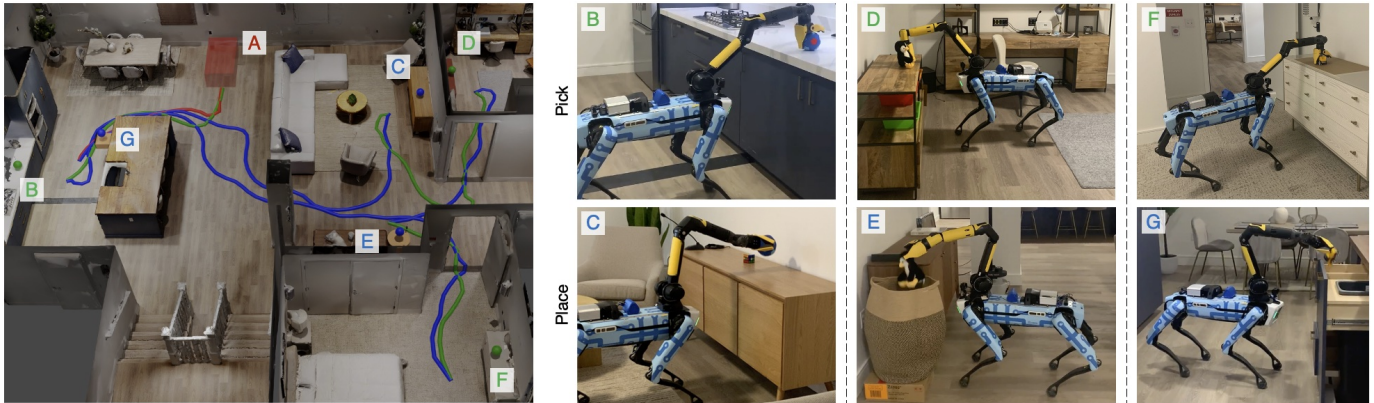[1]Georgia Institute of Technology, [2]FAIR, Meta AI

Fig. 1: Adaptive Skill Coordination (ASC) is deployed on Spot in a novel environment and tasked with mobile pick-and-place, 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 like mobile pick-and-place (*i.e.*, navigating to an object, picking it, navigating to another location, and placing it). ASC consists of three components – (1) a library of basic visuomotor *skills* (navigation, pick, place), (2) a skill *coordination* policy that chooses which skill to use when, and (3) a *corrective* policy that adapts pre-trained skills in out-of-distribution states. All components of ASC rely only on onboard visual and proprioceptive sensing, without requiring detailed maps with obstacle layouts or precise object locations, easing real-world deployment. We train ASC in simulated indoor environments, and deploy it *zero-shot* (without any real-world experience or fine-tuning) on the Boston Dynamics Spot robot in eight novel real-world environments (one apartment, one lab, two microkitchens, two lounges, one office space, one outdoor courtyard). In rigorous quantitative comparisons in two environments, ASC achieves near-perfect performance (59/60 episodes, or 98%), while sequentially executing skills succeeds in only 44/60 (73%) episodes. Extensive perturbation experiments show that ASC is robust to hand-off errors, changes in the environment layout, dynamic obstacles (*e.g.*, people), and unexpected disturbances. Supplementary videos at **adaptiveskillcoordination.github.io**.

Index Terms—AI-Enabled Robotics; Reinforcement Learning; Deep Learning Methods

I. INTRODUCTION

WE study 'in-the-wild' mobile pick-and-place, a subset of the general object rearrangement task [1]. In this task, a robot is initialized in a home and tasked with moving objects from initial to desired locations, emulating 'tidying-up' (Fig. 1). The robot operates entirely using onboard sensors: head- and arm-mounted cameras, proprioceptive joint sensors, and egomotion sensors. It navigates to a receptacle with clutter, like the kitchen counter (whose approximate location w.r.t. the robot's starting pose is known), searches for and picks an object (whose name is known but location, pose, or 3D model are *not* known), navigates to the object's desired place receptacle (with known approximate location), places it, and repeats.

The long-horizon nature of this task makes learning challenging: (1) errors made earlier can limit adequate exploration of states relevant to later parts of the task (*e.g.*, if the robot takes a wrong turn or gets stuck in navigation, it cannot gather experience for picking the target object); (2) conflicting goals for different parts of the task can require careful reward tuning that depends on the current stage of the episode (*e.g.*, whether the robot is now navigating, picking, or placing); and (3) learning to complete the initial stages of the task may make learning different skills needed for later stages challenging (*e.g.*, if the robot learned to avoid obstacles to navigate, it may have difficulty learning to go close to a table for picking).

To tackle these problems, we present Adaptive Skill Coordination (ASC), which consists of three components: (1)

a library of basic visuomotor skills (navigation, pick, and place), (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 in out-of-distribution states. All components of ASC use raw high-dimensional observations (*e.g.*, images) and are trained using reinforcement learning (RL) entirely in the Habitat simulator (Habitat-Sim) [2, 3]. Once trained, it transfers to the real-world 'zero-shot', *i.e.*, without any real-world experience.

We deploy ASC on the Boston Dynamics (BD) Spot robot [4] and conduct experiments in eight diverse and novel real-world environments (one apartment, one lab, two microkitchens, two lounges, one office space, one outdoor courtyard). First, we conduct quantitative comparisons with baselines in two environments – a fully-furnished $185m^2$ apartment, and a $65m^2$ university lab – and find that ASC succeeds at a near-perfect rate (59/60 episodes or 98%), overcoming hardware instabilities, picking failures, and adversarial disturbances like moving obstacles or blocked paths. In comparison, sequentially executing skills only succeeds in 44/60 (73%) episodes, due to hand-off errors and inability to recover from disturbances. Second, we use ReplicaCAD [2] environments in Habitat-Sim to systematically benchmark several ASC ablations and other baselines over 1500 episodes; we find that ASC's coordination policy plays a significant role in avoiding hand-off errors, and the corrective policy further improves performance over other baselines by adapting to out-of-distribution states. Third, we present extensive qualitative evaluations in all eight real-world environments, and demonstrate robustness to dynamic obstacles, adversarial perturbations to the target object, and hardware failures. Finally, we present a direct quantitative comparison of ASC skills against what the BD API provides 'out-of-the-box'. Overall, ASC assumes limited knowledge of the environment and demonstrates robust 'in-the-wild' real-world deployment.

II. RELATED WORKS

Classical task and motion planning (TAMP): Many works study mobile manipulation on humanoid robots [5, 6], wheeled robots [7], and quadrupedal robots [8]. Classical TAMP [9] 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 about the test environments such as pre-built detailed maps with obstacles [7], precise object locations [10], or teleoperation [5, 6, 8]. ASC uses learned skills and does not depend on a planning domain, detailed maps with obstacle layouts, precise object locations, or teleoperation.

Modular learning: [2, 11] use a STRIPS task planner [12] to sequence learned skills; both report hand-off errors as a significant cause of failure, which is a problem we study. [13] and [14] train a coordination policy, but [13] does not scale to cluttered environments (0% success as reported by [14]); [14] assumes access to motion planners and is only demonstrated in simulation. In the real world, [15] learns to pick 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.
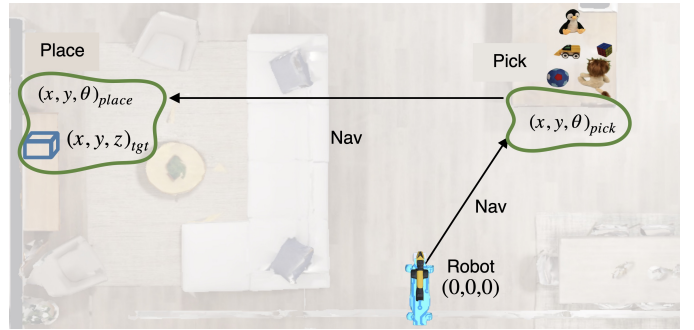

Fig. 2: The robot navigates to a receptacle at $(x, y, \theta)_{pick}$, searches for and picks a target object, navigates to the place receptacle, located at $(x, y, \theta)_{place}$, and places the object at the target place location $(x, y, z)_{tgt}$. Precise object locations and a detailed map of the environment with obstacles are not given.

Fig. 3: ASC is trained entirely within the Habitat simulator. The HM3D dataset is used to train the navigation skill, while ReplicaCAD is used to train the pick, place, coordination, and corrective policies.

SayCan [16] and TidyBot [17] use large language models as task planners and execute basic skills sequentially. However, our experiments show that such sequential skill execution is prone to failure due to hand-off errors. [17] operates in a small area monitored by third-person cameras used to detect and localize objects. In contrast, we rely solely on onboard sensing, which enables easy deployment to new environments.

End-to-end learning: [18]–[23] learn mobile manipulation end-to-end, but with strong assumptions to simplify the task. [18] considers a single corridor with a few obstacles, while [21, 22] do not consider obstacles in the environment. [19, 20] operates in a single simulated room, and assume access to precise object locations. [23] operates in a small area monitored by external cameras to localize objects. ASC operates in large, fully-furnished spaces, with only onboard sensing.

III. TASK: MOBILE PICK-AND-PLACE

The mobile pick-and-place task involves finding target objects and placing them in their desired locations on one of M receptacles, whose approximate locations are known. Receptacles used in our simulation environment include kitchen counters, TV stands, and two types of tables from the ReplicaCAD dataset [2]. Real-world receptacles include various tables, a sofa, a kitchen counter, a drawer, and a hamper, shown in Fig. 1. An approximate receptacle location is defined as a robot pose from which an object or place location on the receptacle is within $30°$ of the robot's heading and 0.6m of its position. In simulation, we sample this location using ground-truth object or place locations, and in the real-world, we manually tele-operate the robot to a receptacle and record a robot pose near the approximate center of the receptacle. We

follow an episodic coordinate system, established by the robot start pose. In 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 relative yaw (Fig. 2). Upon reaching the receptacle, it searches for and picks a target object, then navigates to the object's desired place receptacle approximately located at $(x, y, \theta)_{place}$, placing the object at $(x, y, z)_{tgt}$. A detailed map with obstacle layout or precise object locations is not provided.

Robot observations. The full observation space (Fig. 4) consists of: (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 I_{grip}; (3) a bounding box image I_{bbox} of the target object, if detected by an object detector; (4) the joint angles of the robot arm q_{arm}; and (5) an egomotion sensor estimating the robot's relative heading and displacement from its start pose. The approximate receptacle location where the object can be picked or placed (*e.g.*, counter, hamper) and the place target are also provided.

IV. ADAPTIVE SKILL COORDINATION

Training ASC consists of two steps. First, we train a library of three basic visuomotor skills: navigation, picking, and placing (Fig. 4, left). Next, we train a skill coordination policy that chooses which skills to use when, and a corrective policy that is used instead of the pre-trained skills when out-of-distribution states are perceived (Fig. 4, right). All policies are trained using RL in simulation. In the following section, $d(a, b)$ denotes the geodesic distance (length of the shortest obstacle-free path) between two points a and b; $\Theta(a, b)$ is the angular distance between two orientations a and b; and $\mathbb{I}^{condition} = 1$ if *condition* is true, and 0 if not. $\boldsymbol{p}$ denotes the position and $\boldsymbol{\rho}$ the orientation components of a pose.

A. Basic visuomotor skills

Navigation skill. Spot is initialized in a random location in a scene from the HM3D dataset [24] (Fig. 3), and tasked with reaching within 0.3m and 5° of a goal $(x, y, \theta)_{goal}$, specified relative to its start pose (similar to PointNav [25]). The skill π_{nav} uses the front depth images $I_{front} \in R^{240 \times 212}$, and an egomotion sensor that measures the current pose of the robot's base $(x, y, \theta)_{base}^t$ relative to the start pose, and outputs local forward and angular velocity commands: $v, \omega \sim \pi_{nav}(I_{front}, (x, y, \theta)_{base}^t, (x, y, \theta)_{goal})$, where v and ω have ranges $[-0.5, 0.5]^{m/s}$ and $[-30, 30]^{\circ}/s$. $\boldsymbol{p}_{base}^t$ denotes the 2D position of the robot base $(x, y)_{base}^t$ at time t, and $\boldsymbol{p}_{goal}$ denotes $(x, y)_{goal}$. Δ_t^{nav} denotes a base-distance-to-goal that incorporates both location and heading:

$$\Delta_t^{nav} = d(\boldsymbol{p}_{base}^t, \boldsymbol{p}_{goal})$$
$$+ 0.1 \cdot \begin{cases} \Theta(\theta_{base}^t, \theta_{goal}), & \text{if } d(\boldsymbol{p}_{base}^t, \boldsymbol{p}_{goal}) < 1\text{m} \\ c, & \text{otherwise} \end{cases}$$

where $c = 3.14$ is a constant; Δ_t^{nav} only encourages the robot to orient itself to θ_{goal} when it is within 1m of the goal location. The navigation reward also gives a constant slack penalty of 0.01 (for faster task completion), penalizes

collision, backward movement, and gives a terminal reward if successful:

$$r_{nav,t} = (\Delta_{t-1}^{nav} - \Delta_t^{nav}) - 0.003 \, \mathbb{I}^{collision}$$
$$- 0.003 \, \mathbb{I}^{backward} + 5 \, \mathbb{I}^{success} - 0.01$$

Pick skill. The goal of π_{pick} is to: a) search for the target object, b) center it in the gripper camera view, and then c) pick it. For c), we use the BD grasp API which takes as input a pixel located on the target object. However, our experiments (see Sec. VI-B and videos at adaptiveskillcoordination.github.io) show that the BD grasp API often fails if the target object is far from the gripper ($> \sim 0.8$m), or occluded by other objects; thus, a) and b) are critical, and are the focus of π_{pick}. In simulation, Spot is trained to reach its arm towards an object on one of the receptacles in the ReplicaCAD scene (Fig. 3), starting from random arm joint positions. Multiple distractor objects are placed on the receptacle to simulate clutter and occlusion. π_{pick} receives as input the current joint positions of the arm q_{arm}, and two images: the depth image from the gripper camera $I_{grip} \in R^{240 \times 228}$, and a binary mask indicating which pixels are within the bounding box of the target object $I_{bbox} \in R^{240 \times 228}$. If the target object is not visible, I_{bbox} is all zeros (no bounding box detected). The formulation of I_{bbox} allows the pick skill to generalize to novel objects at test-time, as long as an object detector can provide its bounding box. π_{pick} outputs desired joint angle displacements Δq relative to the current arm joint positions $q_{arm} : \Delta q \sim \pi_{pick}(I_{grip}, I_{bbox}, q_{arm})$. Let $\boldsymbol{p}_{grip}^t$, $\boldsymbol{p}_{obj}$ denote the 3D positions of the gripper and the object; $\boldsymbol{\rho}_{grip}^t$ denote the gripper camera's principle axis; and $\boldsymbol{\rho}_{obj}^t$ denote the unit vector from the gripper camera's center to the target object's center (all are known in sim during training). Analogous to navigation, we define a gripper-distance-to-goal, where the goal is to have the gripper 0.4m away from the target object and pointing directly at it: $\Delta_t^{pick} = |d(\boldsymbol{p}_{grip}^t, \boldsymbol{p}_{obj}) - 0.4| + \Theta(\boldsymbol{\rho}_{grip}^t, \boldsymbol{\rho}_{obj}^t)$. The reward for π_{pick} encourages reducing gripper-distance-to-goal over time, penalizes picking the wrong object, and uses a slack penalty of 0.01 and a terminal success reward:

$$r_{pick,t} = 20 \, (\Delta_t^{pick} - \Delta_{t-1}^{pick}) - 5 \, \mathbb{I}^{wrong\_pick} + 10 \, \mathbb{I}^{success} - 0.01$$

The BD grasp API is called if $\Theta(\boldsymbol{\rho}_{grip}^t, \boldsymbol{\rho}_{obj}^t) < 15°$ (*i.e.*, object is centered) and the object is $0.3 - 0.75$m away from the gripper for 4 consecutive time steps (2 seconds). To determine if these criteria are met, we use the gripper depth image and the center of the detected bounding box to calculate the distance and $\Theta(\boldsymbol{\rho}_{grip}^t, \boldsymbol{\rho}_{obj}^t)$. In simulation, the object is kinematically attached to the gripper, avoiding expensive contact-rich simulation, following [26]. Note, π_{pick} is not given the precise location of the target object (easing real-world deployment), and searches for it with I_{grip} and I_{bbox}.

Place skill. To train the place skill π_{place}, Spot is spawned in front of one of the receptacles in ReplicaCAD (Fig. 3), while holding an object in its gripper. The object must be placed within 0.1m of its desired place location $\boldsymbol{p}_{tgt} = (x, y, z)_{tgt}$, located on the receptacle. One limitation of Spot's embodiment is that the gripper camera is blocked by the gripped object, and front cameras are often blocked by receptacles when Spot is standing close to them; thus, the place skill does
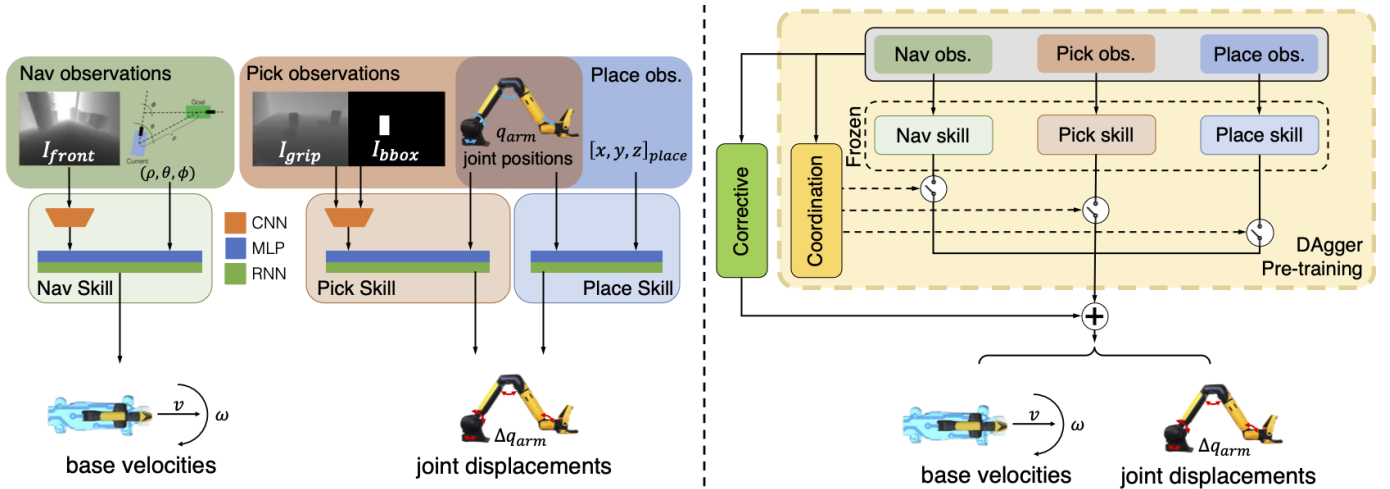
Fig. 4: Training ASC consists of two steps: (left) First, we train a library of 3 basic visuomotor skills in diverse simulated environments. The skills are trained using RL to achieve the relatively shorter-horizon tasks of navigation, picking and placing, and command robot base velocities and joint position deltas. (right) 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 mobile pick-and-place.

not take visual inputs. The output of 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 reward encourages the arm to reach the place target using a distance term, $\Delta_t^{place} = d(\boldsymbol{p}_{grip}^t, \boldsymbol{p}_{tgt})$. It also encourages the gripper to change its orientation to point downwards soon before the object should be released (<0.4m from $(x, y, z)_{tgt}$), so that the object can fall naturally upon opening the gripper, using

$$\Delta_t^{place,\theta} = \begin{cases} \Theta(\boldsymbol{\rho}_{grip}^t, -\vec{z}), & \text{if } d(\boldsymbol{p}_{grip}^t, \boldsymbol{p}_{tgt}) < 0.4\text{m} \\ 3.14, & \text{otherwise} \end{cases}$$

where $\Theta(\boldsymbol{\rho}_{grip}^t, -\vec{z})$ is the angle between the forward axis of the gripper and the negative $\vec{z}$ axis, and $\boldsymbol{p}_{grip}^t$ is the current gripper position. The reward also penalizes collisions with the environment, and uses a success reward and a slack penalty:

$$r_{place,t} = 5 \ (\Delta_{t-1}^{place} - \Delta_t^{place}) - 0.003 \ \Delta_t^{place,\theta}$$
$$- 0.03 \ 25 \ \mathbb{I}^{success} - 0.01$$

In principle, it is possible to replace this 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 stage, we train a mixture-of-experts coordination policy π^{coord} that activates the appropriate (pre-trained and frozen) skills, 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 to such out-of-distribution states.

Skill coordination. Given a library of K pre-trained skills $\Pi = \{\pi_1, \pi_2 \cdots \pi_K\}$, a mixture-of-experts coordination policy π^{coord} learns to coordinate skills based on observations. Let $z_i \sim \pi^z(\pi_i|o)$ be the outputs of a learned gating network π^z representing the probability that skill π_i is selected. We

threshold the *tanh*-activated output of π^z around 0 (*i.e.*, output < 0 is $z_i = 0$, and output > 0 is $z_i = 1$) to obtain a binary selection variable per skill, *i.e.*, $z_i \in \{0, 1\}$, with π_i being selected if $z_i = 1$. Multiple or no skills can be selected at the same time, *i.e.*, $0 \le \sum_i z_i \le K$. 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 $a_{coord} \sim \pi^{coord}(o)$ is:

$$a_{coord} = \sum_{i=1}^{K} z_i \cdot a_i, \quad \text{where } z_i \sim \pi^z(\pi_i|o), \ a_i \sim \pi_i(o_i).$$

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 $\sum_j z_j = 1$ by only selecting the skill that has the highest *tanh-activation*. This creates a hierarchical MoE [27], separating 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, or opening/closing skills, and a learned hierarchical MoE that chooses between them.

Skill correction. Since the skills are trained on simpler tasks (*e.g.*, isolated navigation, picking, placing), there are likely to be states in mobile pick-and-place where they perform poorly. We adapt to such states using a corrective policy π^{corr}, without losing the knowledge gained during the first stage of training. We use the output of the gating network π^z to indicate if a

skill should be adapted or not. Intuitively, π^z picks skills that perform well given observation o and need not be adapted. Conversely, it does not select skills that perform poorly in the current state, and hence should be adapted. Thus, we adapt the action sub-space(s) *unused* by π^z using corrective actions from the corrective policy π^{corr}. Note that π^{corr} adapts to states that are out-of-distribution for the pre-trained skills, but are in-distribution for π^{corr} itself, which is trained on the full mobile pick-and-place task.

If all skills π_j with $a_j \in a_{sub}$ are not selected, the adaptive action $a_{adapt} \in a_{sub}$ using $a_{corr} \sim \pi_{corr}(o)$ is:

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

The final action, $a_{ASC} = a_{coord} + a_{adapt}$, is a sum of the coordination and adaptive actions. a_{ASC} consists of both base and arm actions $(v, \omega, \Delta q)$. The input to both the coordination and corrective policies is the superset of observations used to train the skills (Fig. 4): $o = \{I_{front}, I_{grip}, I_{bbox}, (x, y, \theta)_{goal}, q_{arm}, (x, y, z)_{tgt}\}$. $(x, y, \theta)_{goal}$ is the location of the desired pick or place receptacle, depending on if the robot has picked the object yet. To process the image observations $(I_{front}, I_{grip}, I_{bbox})$, we reuse the trained visual encoders from the basic skills. This enables extraction of useful visual features without re-training the visual encoders. Giving the corrective and coordination policies a superset of visual inputs also enables more efficient mobile pick-and-place; for example, if the coordination policy sees the target object detected in I_{bbox} before fully reaching $(x, y, \theta)_{pick}$, it can terminate navigation early and initiate picking.

Reward. To train the coordination and corrective skills, the robot is tasked with moving one object from one receptacle to a target place location on a different receptacle, using:

$$r_t = 10 \, \mathbb{I}^{\text{success}} - 0.03 \, \mathbb{I}^{\text{collision}} - 0.03 \, \mathbb{I}^{\text{backward}} - 0.003 \, e_t - 0.01$$

where e_t is the sum-of-squares (magnitude) of the actions.

V. TRAINING DETAILS

All policies in ASC are trained in Habitat-Sim using DDPPO [28]. A sparse reward function for training coordination and correction, as described in the previous section, makes RL challenging due to a large state-action space where most action sequences result in suboptimal rewards. To overcome this, we pre-train the coordination policy using the DAgger algorithm [29] with sequential skill-chaining as a teacher (*Seq-skills* in Sec. VI-B). π_z is trained to activate skills in a pre-defined sequence (navigate, pick, navigate, place) based on the robot's observations. The corrective policy, on the other hand, is not warm-started with pre-training due to the absence of labels. Both the coordination and corrective policies are then fine-tuned using RL, resulting in significantly improved performance compared to sequential skill-chaining (experiments in Sec. VI-B). By only incorporating dense rewards for training basic skills and relying on a simple sparse reward for the full task, we minimize the need for extensive RL reward tuning.
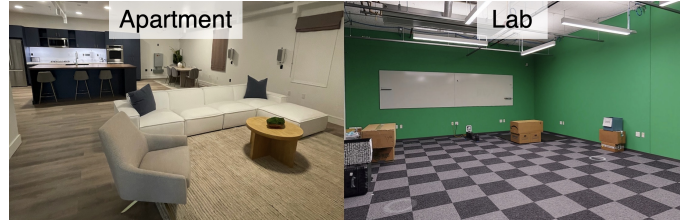

Fig. 5: We present quantitative experiments in an *Apartment* and a *Lab*, 2 of the 8 unseen real-world environments ASC is deployed in.

Policy architecture. Each policy has a 3-layer convolutional neural network (except the place skill), followed by a 2-layer perceptron and a recurrent GRU [30] layer, and finally a fully-connected layer that parameterizes a diagonal Gaussian action distribution. Specifically, a policy π given observations o outputs actions $a \sim \mathcal{N}(\mu, \sigma\mathbf{I})$, where $[\mu, \sigma] = \pi(o)$. Our real-world experiments use a Razer Blade 14 consumer-grade laptop, with an RTX 3070 Max-Q Mobile GPU and an AMD Ryzen 9 CPU; total inference time is about 5 ms.

VI. EXPERIMENTAL EVALUATION

Real-world environments. We deploy ASC on Spot in the real-world, and run quantitative experiments in two environments: a $185m^2$ fully-furnished apartment (*Apartment*) and a $65m^2$ university lab (*Lab*), shown in Fig. 5. 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. Detailed information about these environments, like obstacle layout, size and shape of receptacles, or precise object locations, is not known apriori. All hyperparameters for all approaches are tuned to maximize their performance in a held-out simulation environment. During real-world deployment, these hyperparameters are kept constant.

Robot control. In simulation, we use an articulated Spot URDF model provided by BD [4]. Following [26] for sim-to-real transfer, we use an approximate kinematic simulation of the robot, and depend on low-level hardware controllers in the real world via the BD API. For navigation, 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. Otherwise, the policy receives a collision penalty during training and no movement occurs. Likewise, the arm is moved to the desired joint positions if it would not cause collision. In the real world, the learned policy outputs actions at 2 Hz, which are sent to the BD API, and sensor observations are received from the robot at a frequency of 12 Hz. We also use the BD API to localize the robot and obtain its current pose relative to its start pose. Note that the BD API does not check environment collisions with the arm, and base obstacle avoidance is turned off to allow the robot to move closer to receptacles. Hence, all collision avoidance behavior is learned entirely in simulation.

A. Quantitative experiments in the real world

ASC is tasked with rearranging 30 objects (ranging from a plush penguin/lion/ball, three different toy cars, and a Rubik's

Fig. 6: ASC is deployed in eight real environments (including *Apartment* and *Lab*), showing in-the-wild mobile pick-and-place capabilities.

cube) in the *Apartment* and *Lab*. Mask R-CNN [31] is used for object detection. 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 (Fig. 1). The process is repeated 30 times, to allow the robot to attempt to rearrange each object once. We compare ASC to:

- ASC-NoCorrective: ASC with only coordination and no corrective policy, to study the role of the corrective policy.
- Seq-skills: A skill-chaining baseline that sequentially executes each skill (navigation, pick, navigation, place).
- Seq-skills-premapping: Same as Seq-skills, but the learned navigation skill is replaced with a traditional mapping-based approach. Unlike all other approaches, Seq-skills-premapping receives a pre-built privileged, detailed map of the environment, which is used by a graph-based navigation API provided by BD. This baseline can fail if the layout of the environment changes after mapping [32], which learned navigation is robust to.

Real-world performance. We measure performance as successful mobile pick-and-place episodes, *i.e.*, the number of objects that the robot successfully picked and placed within 0.1m of their target place locations over a total of 60 episodes – 30 in *Apartment* and 30 in *Lab*, over three 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 150m, while in the *Lab* it took about 8-10 minutes and moved about 135m. The robot was completely autonomous throughout all experiments, needing no human intervention. Tab. I shows a quantitative comparison. 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, ASC-NoCorrective rearranged 27/30 in *Apartment* and 27/30 in *Lab*, failing due to the inability of π_{nav} to move close enough to the pick or place receptacle. π_{nav} is trained to avoid obstacles, like receptacles, and hence moving close is out-of-distribution and requires the corrective policy. Seq-skills rearranged 23/30 objects in *Apartment* and 21/30 objects in *Lab*, predominantly failing due to hand-off errors when π_{nav} stopped the robot in a suboptimal position for subsequent picking or placing. Seq-skills-premapping rearranged 22/30 objects in the *Apartment*, and 21/30 in the *Lab*, also failing due to hand-off errors, and once due to a navigation error, where the robot became stuck and could not navigate to the target location. Seq-skills-premapping performed similar to Seq-skills, indicating that hand-off errors were not eliminated by utilizing a classical map-based navigation method instead of learned navigation.

Row #		Apartment (real) 30 episodes	Lab (real) 30 episodes	Simulation 1500 episodes
1	ASC	**96.7%**$_{\pm 3.3}$	**100.0%**$_{\pm 0.0}$	**94.9%**$_{\pm 0.6}$
2	ASC-NoCorrective	90.0%$_{\pm 5.5}$	90.0%$_{\pm 5.5}$	92.3%$_{\pm 0.7}$
3	Seq-skills	76.7%$_{\pm 7.7}$	70.0%$_{\pm 8.4}$	87.8%$_{\pm 0.8}$
4	Seq-skills-premapping	73.3%$_{\pm 8.1}$	70.0%$_{\pm 8.4}$	-
5	ASC-NoCorrective-Finetune	-	-	43.9%$_{\pm 1.3}$
6	ASC-NoDAgger	-	-	5.6%$_{\pm 0.6}$
7	Mono-RL	-	-	35.8%$_{\pm 1.2}$

TABLE I: We evaluate ASC on mobile pick-and-place in *Apartment*, *Lab*, and simulation, comparing ASC against various baselines and ablations to highlight the significance of each component of ASC.

Both performed worse than ASC, which is robust to hand-off errors and re-invoked the navigation skill or used the corrective policy to move the robot if needed.

In-the-wild mobile pick-and-place. We also deploy ASC in 6 additional real-world environments (see Fig. 6), highlighting its robustness at solving mobile pick-and-place in-the-wild. We use Owl-ViT [33] to detect a wide range of objects. Videos can be found at adaptiveskillcoordination.github.io.

B. Mobile pick-and-place in simulation

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

- Fine-tuning: An ablation of ASC that fine-tunes the 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.
- ASC-NoDAgger: An ablation of ASC which does not pre-train the coordination policy using DAgger.

Tab. I shows a quantitative comparison over 1500 mobile pick-and-place episodes in 79 different furniture layouts in ReplicaCAD (mean success and standard error). Learned skill coordination and correction in ASC outperforms Seq-skills (94.9%$\pm$0.6 versus 87.8%$\pm$0.8), by avoiding hand-off errors in pick/place. ASC-NoCorrective is unable to generalize to states which are out-of-distribution for pre-trained skills, and hence performs slightly worse than ASC (92.3%$\pm$0.7 for No-corrective versus 94.9%$\pm$0.6 with ASC). ASC-NoCorrective-Finetune suffers from 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 sparsity of the reward function (35.8% $\pm$ 1.2 using Mono-RL). This is consistent with findings in [2] that monolithic RL needs
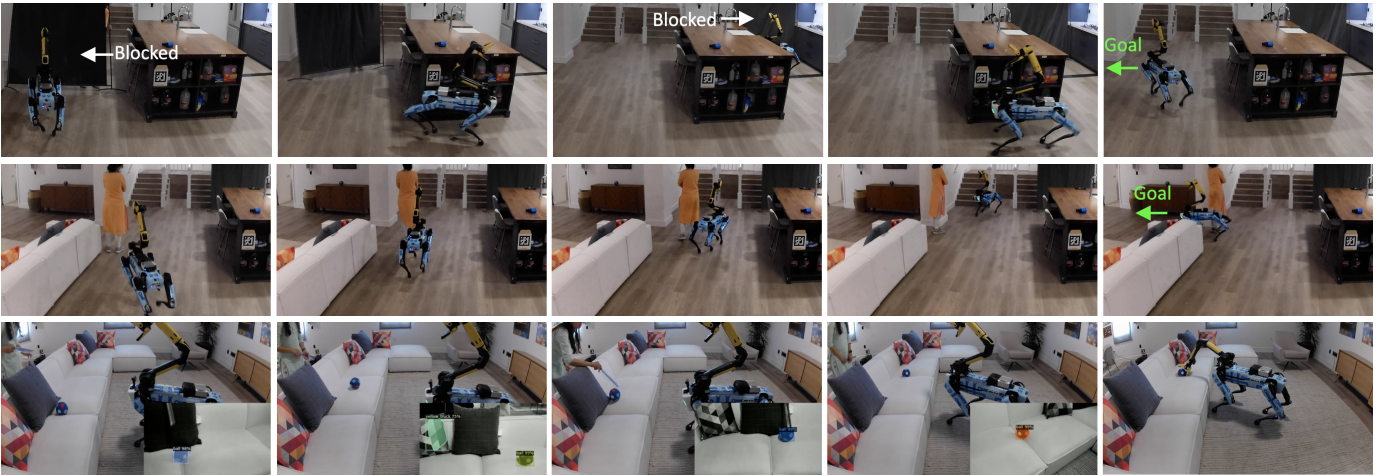
Fig. 7: ASC is robust to several types of real-world disturbances (see video at adaptiveskillcoordination.github.io). *Top*: ASC can reroute the robot upon sudden changes to the environment layout blocking its path, until it successfully finds a path to the goal, even if that path was previously blocked. *Bottom*: Since ASC does not take object positions as input, it is also robust to perturbations like moving objects.

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

TABLE II: The BD API can navigate to unobstructed goals (*Easy Nav*), but fails to circumvent obstacles (*Medium Nav*) or exit a room (*Difficult Nav*). It grasped 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 three navigation episodes per difficulty, and 18 pick episodes in the *Lab*.

well-tuned rewards to learn complex tasks. ASC-NoDAgger performs the worst of all baselines ($5.6\% \pm 1.0$), showing the importance of pre-training the coordination policy with DAgger when using sparse rewards. Each component of ASC is important for robust performance. Skill coordination outperforms skill sequencing, and the corrective policy adds robustness by adapting to states that are out-of-distribution for the pre-trained skills without catastrophic forgetting.

Coordination vs. corrective policy. In Tab. I, we see that adding the coordination policy to Seq-skills improves performance significantly (row 3→2, ~12.6%↑ across columns), and adding the corrective policy leads to further improvement (row 2→1, ~6.4%↑ across columns). Learning a coordination policy enables better hand-off between navigation and pick/place skills, over sequentially executing skills. The corrective skill further improves performance by adapting to out-of-distribution states, not seen during basic visuomotor skill training. For example, the corrective policy can move the robot away from the navigation target to pick/place an object.

C. Robustness to perturbations

An assistive robot should be robust to perturbations such as a human walking in front of it, or slight movements of the object that it is picking. Since ASC uses reactive visuomotor skills (and is not reliant on an outdated map), it is robust to such disturbances (see Fig. 7). When faced with changes in environment layout or dynamic obstacles, ASC re-routes the robot to a new collision-free path to the goal. Since ASC

does not take the object position as input, it is also robust to movements of the target object. If the object is moved while the robot is searching, ASC continues searching for the object. On hardware, sometimes the robot becomes unstable when picking, prompting the BD API to move the robot away from the object to re-gain balance. Such cases are out-of-distribution, as ASC is not trained on such disturbances, but ASC re-activates π_{nav} and moves the robot back towards the object, before picking the target object. In these cases, the importance of the coordination policy is evident, as it considers the most recent observations to reactively decide whether to re-invoke navigation or pick/place skills. On the other hand, Seq-skills and Seq-skills-premapping are not robust to such disturbances, as they follow a pre-defined sequence of skills, and do not take current observations into account when deciding which skill to activate, ultimately failing to pick the target object. Robustness to such disturbances is achieved without any explicit training, emerging naturally from the design choices made in ASC.

D. Comparing ASC skills and the Boston Dynamics API

Boston Dynamics provides navigation, manipulation, and grasping APIs on Spot, which serve as the backbone for ASC. This has two advantages: ASC is able to leverage controllers specifically designed and tuned for Spot by BD, and 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 without a map. As shown in Tab. II, in the real world, 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*). For such challenging goals, the

BD API requires a map of the environment, but can fail if layout of the environment changes [32]. In contrast, the learned navigation skill can perform long-range navigation on Spot in all cases, and is robust to changes in environment layout.

The BD grasp API typically fails to grasp the target object if it is partially occluded, as it may opt to grasp the occluding object 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 API. Tab. II 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, despite ASC's enhancements, the BD grasp may sometimes fail to pick an object due to hardware instability or camera latency. In such cases, ASC re-activates the pick skill, which searches for the object, and re-grasps.

VII. LIMITATIONS AND CONCLUSION

In this work, we present Adaptive Skill Coordination (ASC), an approach that coordinates pre-trained skills and adapts them based on observations. ASC is deployed zero-shot on the Spot robot for mobile pick-and-place in eight unseen real-world environments, and achieves near-perfect performance in quantitative experiments in two environments, without the need for detailed maps with obstacle layouts or precise object locations. In comparison, sequential skill-chaining suffers from hand-off errors and inability to recover from disturbances. Despite being trained entirely in simulation, ASC is robust to real-world disturbances such as dynamic obstacles, changes to environment layout, and target objects being moved mid-episode.

We see several limitations and areas for future work. First, while ASC is a reasonably general approach, our experiments are limited to mobile pick-and-place of small objects from open receptacles. Future work involves studying generalization to a larger number of skills in order to complete more complex mobile manipulation. Second, while our task and goal specification is consistent with community benchmarks [1, 34], it can be improved by allowing goals to be specified with natural language instead. Finally, our experiments are conducted in static environments where the robot is the only active agent; future work involves studying assistive mobile manipulation in multi-agent and dynamic environments.

REFERENCES

[1] D. Batra *et al.*, "Rearrangement: A challenge for embodied AI," *arXiv preprint arXiv:2011.01975*, 2020.
[2] A. Szot *et al.*, "Habitat 2.0: Training home assistants to rearrange their habitat," in *NeurIPS*, 2021.
[3] M. Savva *et al.*, "Habitat: A Platform for Embodied AI Research," in *ICCV*, 2019.
[4] "Boston dynamics," https://www.bostondynamics.com/spot.
[5] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization based full body control for the atlas robot," in *Humanoids*. IEEE, 2014.
[6] S. Kuindersma *et al.*, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, no. 3, pp. 429–455, 2016.
[7] 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.

[8] Y. Ma, F. Farshidian, T. Miki, J. Lee, and M. Hutter, "Combining learning-based locomotion policy with model-based manipulation for legged mobile manipulators," *IEEE RA-L*, pp. 2377–2384, 2022.
[9] 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, 2021.
[10] 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 *ICRA*. IEEE, 2020, pp. 5678–5684.
[11] J. Gu, D. S. Chaplot, H. Su, and J. Malik, "Multi-skill mobile manipulation for object rearrangement," *arXiv preprint arXiv:2209.02778*, 2022.
[12] 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.
[13] 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.
[14] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation," *arXiv preprint arXiv:2008.07792*, 2020.
[15] 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 *CoRL*, 2022.
[16] M. Ahn *et al.*, "Do as I can, not as I say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.
[17] J. Wu *et al.*, "Tidybot: Personalized robot assistance with large language models," *arXiv:2305.05658*, 2023.
[18] 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:2003.02637*, Feb. 2020.
[19] T. Ni, K. Ehsani, L. Weihs, and J. Salvador, "Towards disturbance-free visual mobile manipulation," in *WACV*, 2023, pp. 5219–5231.
[20] K. Ehsani *et al.*, "Manipulathor: A framework for visual object manipulation," in *CVPR*, 2021, pp. 4497–4506.
[21] 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, 2021.
[22] 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.
[23] S. Jauhri, J. Peters, and G. Chalvatzaki, "Robot learning of mobile manipulation with reachability behavior priors," *IEEE RA-L*, 2022.
[24] S. K. Ramakrishnan *et al.*, "HM3D: 1000 large-scale 3D environments for embodied AI," in *NeurIPS*, 2021.
[25] P. Anderson *et al.*, "On evaluation of embodied navigation agents," *arXiv:1807.06757*, 2018.
[26] 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 *CoRL*, 2022.
[27] 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.
[28] E. Wijmans *et al.*, "DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames," in *ICLR*, 2020.
[29] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011, pp. 627–635.
[30] 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.
[31] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.
[32] "GraphNav Service - Spot 3.2.1 documentation," https://dev.bostondynamics.com/docs/concepts/autonomy/graphnav_service.
[33] M. Minderer *et al.*, "Simple Open-Vocabulary Object Detection," in *ECCV*, 2022.
[34] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2020.