This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Is Mapping Necessary for Realistic PointGoal Navigation?

Ruslan Partsey[1*] Erik Wijmans[2,3] Naoki Yokoyama[2] Oles Dobosevych[1] Dhruv Batra[2,3] Oleksandr Maksymets[3]
[1]Ukrainian Catholic University [2]Georgia Institute of Technology [3]Meta AI
rpartsey.github.io/pointgoalnav

Abstract

Can an autonomous agent navigate in a new environment without building an explicit map?

For the task of PointGoal navigation ('Go to Δx, Δy'*) under idealized settings (no RGB-D and actuation noise, perfect GPS+Compass), the answer is a clear 'yes' – map-less neural models composed of task-agnostic components (CNNs and RNNs) trained with large-scale reinforcement learning achieve 100% Success on a standard dataset (Gibson [24]). However, for PointNav in a* realistic *setting (RGB-D and actuation noise, no GPS+Compass), this is an open question; one we tackle in this paper. The strongest published result for this task is 71.7% Success [39].*[1]

First, we identify the main (perhaps, only) cause of the drop in performance: absence of GPS+Compass. An agent with perfect GPS+Compass faced with RGB-D sensing and actuation noise achieves 99.8% Success (Gibson-v2 val). This suggests that (to paraphrase a meme) robust visual odometry is all we need for realistic PointNav; if we can achieve that, we can ignore the sensing and actuation noise.

With that as our operating hypothesis, we scale dataset size, model size, and develop human-annotation-free data-augmentation techniques to train neural models for visual odometry. We advance state of the art on the Habitat Realistic PointNav Challenge – SPL by 40% (relative), 53 to 74, and Success by 31% (relative), 71 to 94. While our approach does not saturate or 'solve' this dataset, this strong improvement combined with promising zero-shot sim2real transfer (to a LoCoBot robot) provides evidence consistent with the hypothesis that explicit mapping may not be necessary for navigation, even in a realistic setting.

1. Introduction

The ability to navigate in a novel environment solely from egocentric perception is an essential requirement for building intelligent and helpful robots. To make progress on this long-term vision, the task of PointGoal navigation (PointNav) [1], *i.e.* asking a robot to *'go to* $(\Delta x, \Delta y)$*'* relative to its starting location), has become a core task.

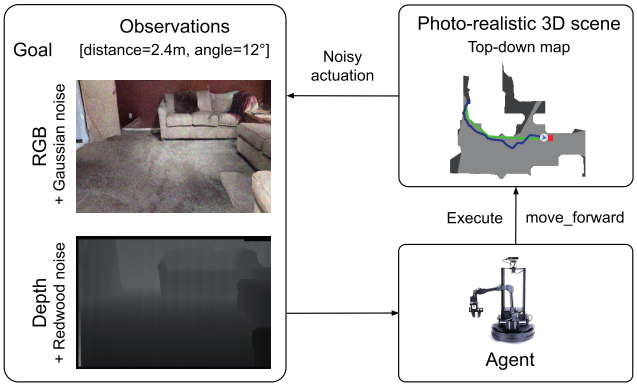

Figure 1. **PointNav.** The agent is tasked with navigating from its starting location (blue square) to a goal location (red square) specified as coordinates relative to its initial place. It must do so from solely an RGB-D camera. The blue line shows the agent's trajectory, and the green line indicates the oracle path.

We are interested in the question – can an agent navigate in a new environment without building an explicit map?[2]

This question is of deep scientific interest. Decades of research in intelligent animal navigation show that various animals build 'cognitive maps' [21, 31] of their environment. For decades, robotics research has treated explicit mapping and localization [2, 20, 27, 30] as integral components in a navigation robot. There are many good reasons to develop mapping technology, but we simply don't know whether mapping is *necessary* for navigation. One way to resolve this is to refute the contrapositive – if we demonstrate a map-less approach that can navigate, that will imply that explicit mapping is not necessary for successful navigation.

Under idealized settings – perfect localization via a noise-free `GPS+Compass` sensor, egocentric sensing via

*Correspondence to partsey@ucu.edu.ua

[1]According to Habitat Challenge 2020 PointNav benchmark held annually. A concurrent as-yet-unpublished result has reported 91% Success on 2021's benchmark, but we are unable to comment on the details because an associated report is not available.

[2]We distinguish an explicit mapping mechanism from implicit spatial understanding that may emerge from building task-specific end-to-end-learned representations. The former is designed, the latter is emergent.

a noise-free RGB-D sensor, and absence of any actuation noise – map-less navigation models composed of task-agnostic neural components (CNNs and RNNs) trained with large-scale reinforcement learning achieve 100% Success [24, 33] on a standard dataset (Gibson [35]). However, under *realistic* settings – where the agent must self-localize (*i.e.* no `GPS+Compass` sensor), and must contend with RGB-D sensing noise and actuation noise – this is an open question; one we tackle in this paper. The strongest published result for this task is 71.7% Success [39].

To make systematic progress, we first study a simpler version of the realistic setting where the agent is given ground-truth `GPS+Compass`, isolating localization difficulties from the ability to deal with noisy perception and control. While prior work in this setting [39] reported fairly a high success rate (97%), we significantly sharpen this and demonstrate near-perfect performance again (99.8% Success rate on Gibson val split). Our results leave no room for doubt and confirm that the *only* performance-limiting factor is the agent's ability to self-localize.

With this limiting-factor identified, we study the localization, or visual odometry (VO) module. It takes as input two successive observations O_{t-1} and O_t and outputs the relative pose change $(\Delta x, \Delta y, \Delta z, \Delta\theta)$, that is then used to update the location of the goal relative to the robot, which is consumed by the navigation policy.

We present a series of broadly-applicable modifications that improve agent navigation performance considerably, from 64% Success/52% SPL to 96% Success/77% SPL on the Habitat Challenge 2021 setting. These modifications are all motivated by the need for robust visual odometry in service of navigation, specifically:

1. **Action conditioning via action embeddings.** It is important to recognize that our goal is not visual odometry in isolation but in the context of navigation. Specifically, we know what action (`move_forward` $0.25m$, `turn_left` 30° or `turn_right` 30°) was executed and should use this information; this observation is not new and has been made in prior work [39]. We find that converting a 1-hot representation of the actions into continuous embeddings and concatenating them to the last two fully-connected layers in the VO network significantly improves performance by +8 Success/+5 SPL.
2. **Training-time data augmentation.** Data augmentation is one of the most successful methods for regularizing learning techniques [17, 37, 38]. We construct navigation- and odometry-specific augmentations – *e.g.* when an agent rotates in-place to produce observations O_{t-1} and O_t, we can create a new training image that relates O_t and O_{t-1} via the inverse pose and turning action. We also propose a new augmentation called `Flip` (described in Sec. 4.2). Cumulatively, they improve performance by +2 Success/+1 SPL.
3. **Test-time data augmentation for ensembling.** To improve robustness we perform all augmentations at test-time and aggregate predictions across all combinations. This improves performance by +3 Success/+3 SPL.
4. **Increased dataset size and model size.** Finally, we study the effects of increased dataset size from 500k to 1.5M observation pairs (+8 Success/+7 SPL), larger model size (+3 Success/+3 SPL), and a further dataset increase from 1.5M to 5M (+8 Success/+6 SPL).

2. Preliminaries: PointGoal Navigation

In PointNav (illustrated in Fig. 1), an agent is initialized in previously unseen environment and is tasked to reach the goal specified relative to its starting location. The action space is discrete and consists of four types of actions: `stop` (to end the episode), `move_forward` by $0.25m$, `turn_left` and `turn_right` by angle α[3].

The agent is evaluated via three primary metrics. 1) Success, S_i, where an episode i is considered successful if the agent issues the `stop` command within $0.36m$ ($2\times$agent radius) of the goal. 2) Success weight by (inverse normalized) Path Length (SPL) [1], where success is weighted by the efficiency of the agent's path. Formally, for episode i, let S_i be a binary indicator of success, p_i be the length of the agent's path, and l_i be the length of the shortest path (geodesic distance), then for N episodes

$$\text{SPL} = \frac{1}{N}\sum_{i=1}^{N} S_i \cdot \frac{l_i}{\max(p_i, l_i)}. \tag{1}$$

3) SoftSPL [9], where binary success is replaced by progress towards goal. Formally, for episode i, let d_{0_i} be the initial distance to goal and d_{T_i} be the distance to goal at the end of the episode (on both successes and failures), then

$$\text{SoftSPL} = \frac{1}{N}\sum_{i=1}^{N} \left(1 - \frac{d_{T_i}}{d_{0_i}}\right) \left(\frac{l_i}{\max(p_i, l_i)}\right). \tag{2}$$

Embodiment. Driven by experiments in reality the agent's specification matches the LoCoBot's [4] specification. The agent is equipped with an RGB-D camera mounted at a height of $0.88m$ and tilted -20° (angled downwards towards the floor; pitch or camera azimuth angle). Camera's resolution is 360×640 pixels with 70° horizontal field of view. Base radius is 0.18m.

2.1. PointNav-v1: Idealized (Noise-less) Setting

In idealized setting (named 'v1'), the agent was equipped with noise-free RGB-D camera, given access to ground-truth localization (via an `GPS+Compass` sensor), and movement was deterministic/noise-free (meaning `turn_right` 10° always turned the agent *exactly* 10°).

[3] In PointNav-v1 $\alpha = 10^\circ$, in PointNav-v2 $\alpha = 30^\circ$.

[4] LoCoBot is a low-cost mobile manipulator suitable for both navigation and manipulation (http://www.LoCoBot.org).

The agent could also 'slide' along walls – a commonplace behavior in video games that improves human control but was later found to degrade sim-to-real performance [14].

State-of-the-art approaches for this task rely on large-scale reinforcement learning and have begun to saturate the available datasets: *e.g.* Wijmans *et al.* [33] reported 99% Success/94% SPL on Gibson test, Ramakrishnan *et al.* [24] sharpened this result to 100% Success/94% SPL on Gibson test, 94% Success/83 SPL on MP3D test, and 99% Success/92% SPL on HM3D test. Overall, PointNav-v1 is largely considered satured or 'solved'.

2.2. PointNav-v2: Realistic (Noisy) Setting

Noiseless sensing and actuation simply do not yet exist. Different lightning conditions, surface properties (such as friction), and other sources of error cause actuation and sensing noise that introduce significant drift over a long trajectory. Moreover, high-precision localization in indoor environments can not be assumed in realistic settings.

The so-called 'realistic' (or v2) setting of PointNav addresses these shortcomings of the v1 by introducing actuation noise (modeled by benchmarking the LoCoBot robot [19]), removing `GPS+Compass`, and adding noise to the RGB-D camera. To simulate real-world camera RGB and Depth, noise models from [8] were used (Gaussian noise model for RGB and Redwood noise model for Depth).

Initial attempts to directly apply PointNav-v1 techniques to PointNav-v2 were largely unsuccessful (≈5% Success [9]). More recent methods [9, 39] train a navigation policy with access to ground-truth localization and then replace ground-truth localization with estimated localization by integrating the egomotion estimates from a visual odometry module. The strongest published result for this task is 71.7% Success and 53% SPL [39], indicating that navigation with noisy actuation and sensing continues to be an open frontier for research.

3. Navigation Policy

Our pipeline consists of two components: a navigation policy (nav-policy) that given observations O_t at time step t decides which action to take to reach the goal and a visual odometry (VO) module that given two consecutive observations (O_{t-1}, O_t) estimates relative pose change (egomotion) that is further used to update the goal coordinates after each step (see Fig. 2). This decoupling of roles is a natural choice. It builds upon the results in the idealized setting that has been used in prior work and is used in the previous state of the art [9, 39]. In this section we describe our navigation policy and show that it is capable of near-perfect navigation with noisy actuations and noisy RGB-D sensing when given ground-truth localization, demonstrating that visual odometry is the bottleneck. We describe the details of our visual odometry module in the next section (Sec. 4). We use the Habitat platform [25] to simulate navigation experiments.

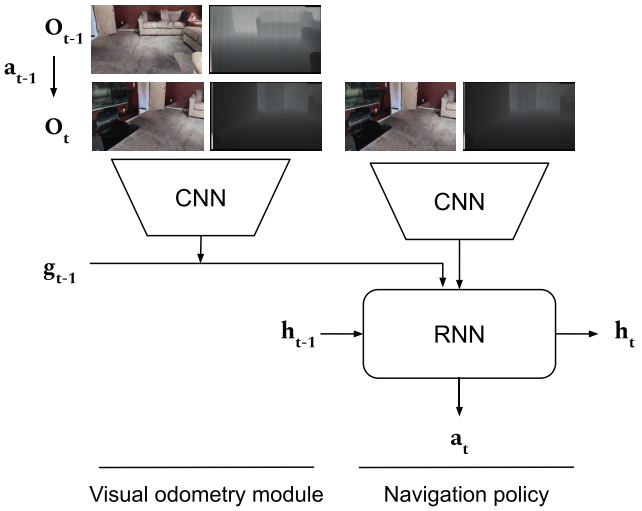

Figure 2. **Agent architecture for Realistic PointGoal navigation** consisting of an RNN-based RL navigation policy and CNN-based visual odometry (VO) module. Inputs are g_{t-1} - goal coordinates wrt. previous pose, a_{t-1} - previous action, O_{t-1} - observations at previous timestep, and O_t - current observations. First, VO predicts the change between $t-1$ and t and then update the goal to be wrt. current pose. The updated goal location is given to the navigation policy along with O_t to predict the next action a_t. The initial goal location estimate is equal to ground truth goal location (as per the task specification).

3.1. Architecture

We train the navigation assuming perfect odometry (use ground-truth localization provided by `GPS+Compass` sensor) and then use VO module estimates as a drop-in replacement of ground-truth localization sensor without fine-tuning (idea introduced by Datta *et al.* [9]; also used by Zhao *et al.* [39]). This also allows us to evaluate the policy's performance in isolation of the visual odometry module.

We use the same navigation policy as Wijmans *et al.* [33], consisting of a two-layer Long Short-Term Memory (LSTM) [12] and a half-width ResNet50 [11] encoder. At each timestep, the policy is given the output from the noisy `Depth` sensor (following common practice for the navigation policy, we discard `RGB`) and idealistic `GPS+Compass` sensor (ground-truth localization, that is replaced by the visual odometry estimates during evaluation). Before passing through the feature encoder, visual observations are transformed using `ResizeShortestEdge` and `CenterCrop` observation transforms; the former resizes the shortest edge of the input to 256 pixels while maintaining aspect ratio, the latter center crops the input to 256 × 256 pixels.

3.2. Training Details

We use the train split of the full Gibson data (scans with ratings of 0 or higher) [35]. We leverage Decentralized Distributed Proximal Policy Optimization (DD-PPO) [33] and

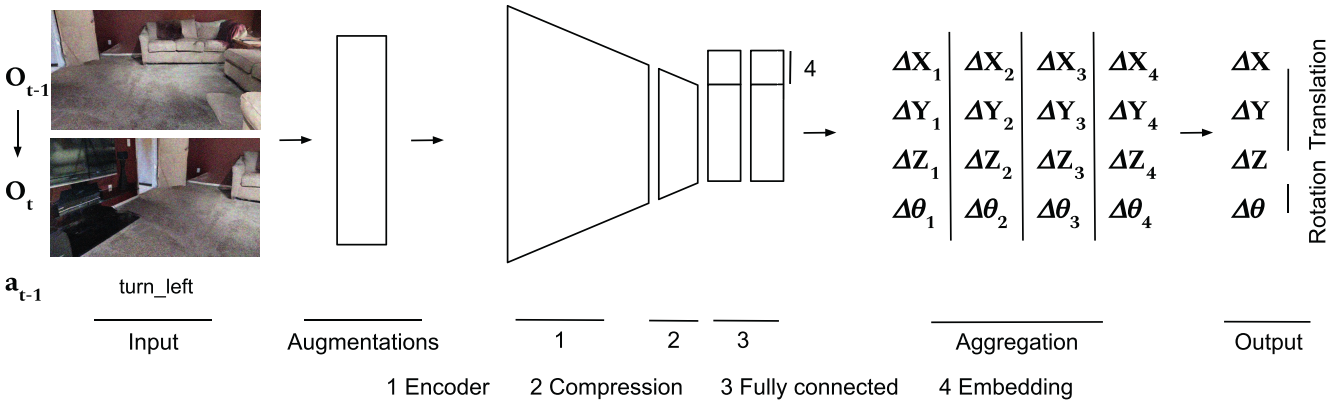

Figure 3. **Visual odometry module.** At inference the input pair of observations (O_{t-1}, O_t) is transformed by applying `Swap` and `Flip` augmentations. In total the visual odometry model receives two observation pairs for `move_forward` (original and flipped) and four observation pairs for `turn_{left,right}` actions (original, flipped, swapped(original), swapped(flipped)). In the aggregation stage outputs are transformed back to original coordinate frame by applying the inverse transformation for each augmentation and then averaging to produce the final egomotion estimate (details in Sec. 4.3).

Wijmans *et al.*'s reward structure to train the policy.

For an episode i, the agent receives a 'terminal' reward of $r_T = 2.5 \cdot \text{Success}_i$ ($r_T = 2.5 \cdot \text{SPL}_i$ in later experiments) that encourages it to stop at the correct location (and take an efficient path), and a shaped reward $r_t(a_t, s_t) = -\Delta_{\text{geo\_dist}} - 0.01$, that encourages it to take steps towards the goal (while being efficient), where $\Delta_{\text{geo\_dist}}$ is the change in geodesic distance to the goal by performing action a_t in state s_t. Note that reward is not available at test-time. We train with 64 GPUs (workers). Throughout our experimentation we never discarded the weights of a trained navigation policy. We trained for 2.5 billion steps on Gibson 4+, then for another 2.5 billion steps on Gibson 0+, and finally for another 2.5 billion steps on Gibson 0+ with the terminal reward weighted by SPL. We started each stage with the best (by val performance) agent from the previous stage.

3.3. Ground-Truth Localization Performance

To isolate the performance of the navigation policy from the visual odometry module, we examine performance of our agent with access to ground-truth `GPS+Compass`. On the Gibson val dataset, our agent achieves 99.8% Success and 80 %SPL in the PointNav-v2 setting. This result shows that near-perfect success, even with noisy observations and actuations, is achievable without building an explicit map.

To answer if near-perfect SPL is also achievable, we need a tight upper-bound on SPL in the realistic setting. Recall that in the realistic setting actuations are noisy. Thus, even an oracle agent with full knowledge of the environment may not be able to follow the shortest path and achieve 100% SPL. This particular problem is exacerbated because 'sliding' is disabled, meaning that if an agent is traveling close to an obstacle (as shortest path typically do), noisy actuation may bring it into contact with the obstacle, requiring backtracking or dislodging and adding to its path length.

To determine a tighter upper-bound on SPL, we implement a heuristic planner that uses the ground-truth map to choose motion primitives (`turn_{left,right}` ×N, then `move_forward`). The planner selects the primitive that best reduces distance to goal using the ground-truth geodesic distance (thereby using the ground-truth map), executes the first action in the selected primitive, and then re-runs the selection process until the goal is reached. On Gibson validation, the oracle achieves 84% SPL. Thus, we shouldn't expect 100% SPL in the realistic setting.

We then further tighten the upper-bound by accounting for the privileged information (the ground-truth map) given to the oracle. Consider the idealized setting, in this setting the challenge for the agent is path-planning in unknown environments, not additionally contenting with noisy actuations and observations. This setting is also considered 'solved' on the Gibson dataset, making it an ideal setting the quantify the impact of the ground-truth map. In the idealized setting, on Gibson val, the oracle achieves 99% SPL while the best known result for a learned agent is 97% SPL [33]. Using either the absolute or relative difference, we would expect approximately 82% SPL to be achievable by a learned agent in the realistic setting as the oracle achieves 84% SPL. While 80% is not quite 82%, this indicates that visual odometry is the limiting factor (the best result with visual odometry is 63% SPL) and we turn our focus towards this component for the rest of the paper.

4. Visual Odometry

The visual odometry model takes a pair of 180×360 RGB-D frames stacked channel-wise as input and predicts the relative pose change between the two camera positions, $\Delta_{\text{pose}} = (\Delta x, \Delta y, \Delta z, \Delta\theta)$, where $\Delta x, \Delta y, \Delta z$ refer to

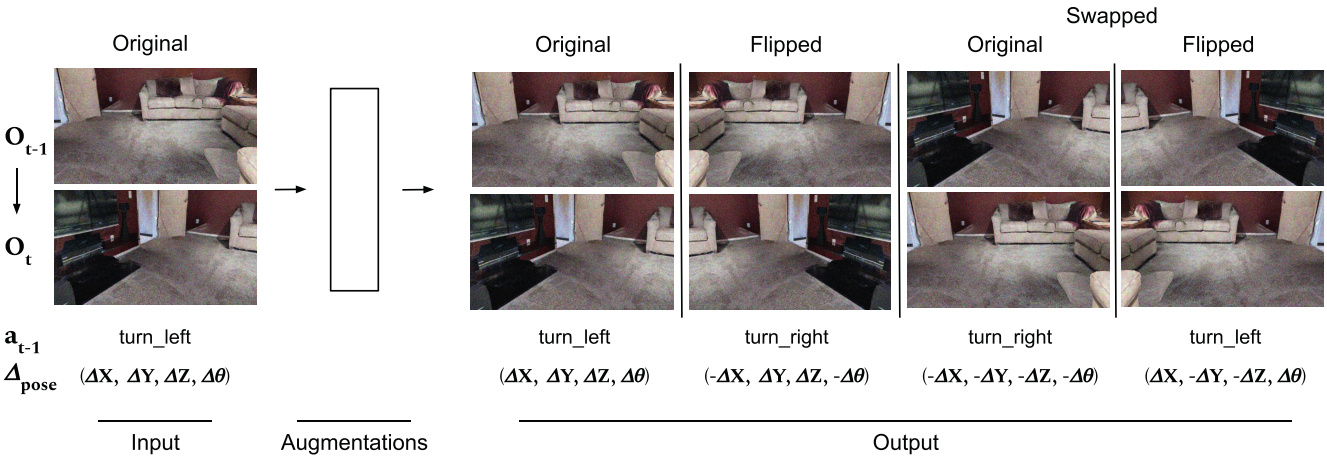

Figure 4. **Augmentations.** An input pair of observations (O_{t-1}, O_t) are transformed by applying `Swap` (returns (O_t, O_{t-1}) and `Flip` (flips observations about their vertical axis) augmentations, producing two transformed pairs as output for `move_forward` action (original and flipped) and four transformed pairs as output for `turn_{left,right}` actions (original, flipped, swapped(original), swapped(flipped)) (details in Sec. 4.2).

the 3D translation of the camera center and $\Delta\theta$ refers to the rotation about the gravity vector ('yaw' or robot heading).

The visual odometry model is represented as ResNet [11] encoder followed by a compression block and two fully connected (FC) layers. We replace BatchNorm [13] with GroupNorm [34] (we found it to work better) and use half of the width. The compression block consists of 3×3 Conv2d+GroupNorm+ReLU. We apply DropOut [28] with 0.2 probability between fully connected layers. Full VO pipeline is illustrated in Fig. 3.

4.1. Action Embedding

Actuation noise and collisions affect agent translation and rotation for each action type differently (the agent may rotate while moving forward and move while rotating in place [19]). This motivated us to study incorporating knowledge of the action taken between two consecutive observations as an additional input. We represent the action taken as an embedding – *fixed* action-specific vector of length 16 that is concatenated to the flattened output from the feature encoder. We *do not* apply a DropOut to action embedding as we find this harms performance. To further increase the importance of the action, we concatenated the embedding to the input of all fully connected layers.

4.2. Train-Time Augmentations

Given a pair of observations, (O_{t-1}, O_t), the action taken between them, a_{t-1}, and their relative change in pose Δ_{pose}, we use the following augmentations.

`Swap`. For every training tuple $(O_{t-1}, O_t, a_{t-1}, \Delta_{\text{pose}})$ where a_{t-1} is a rotation action we create an extra training example $(O_t, O_{t-1}, a_{t-1}^{\texttt{Swap}}, \Delta_{\text{pose}}^{\texttt{Swap}})$ where $a_{t-1}^{\texttt{Swap}}$ is the effective action taken (`turn_left` → `turn_right` and `turn_right` → `turn_left`) and $\Delta_{\text{pose}}^{\texttt{Swap}}$ is the change in pose after swapping (negation of all components). As in Zhao *et al*. [39], this augmentation leverages the order invariance of `turn_{left,right}` actions.

`Flip`. In architecture and indoor design, it is common to prepare mirror-image floor plans (*e.g*. kitchen on the left, living on the right and vice versa) to increase the number of options. As shown in Fig. 4, we simulate the robot navigating in a mirror-image house by flipping its camera image about the vertical axis. Specifically, for every training tuple $(O_{t-1}, O_t, a_{t-1}, \Delta_{\text{pose}})$ we generate an additional training example $(O_{t-1}^{\texttt{Flip}}, O_t^{\texttt{Flip}}, a_{t-1}^{\texttt{Flip}}, \Delta_{\text{pose}}^{\texttt{Flip}})$, where $O_{t-1}^{\texttt{Flip}}$, $O_t^{\texttt{Flip}}$ are the RGB-D observations flipped along their vertical axis, $a_{t-1}^{\texttt{Flip}}$ is the effective action after the flip (`turn_left` → `turn_right`, `turn_right` → `turn_left`, and `move_forward` remains the same), and $\Delta_{\text{pose}}^{\texttt{Flip}} = (-\Delta x, \Delta y, \Delta z, -\Delta\theta)$.

We also apply the composition of `Flip` and `Swap` (similar to `torchvision.transforms.Compose` [22]). Note that `Flip` then `Swap` is the same as `Swap` then `Flip`. All four combinations of Flip and Swap are shown in Fig. 4.

4.3. Test-Time Augmentations

We adapt the common practice of test-time augmentation in image classification to visual odometry. Specifically, we apply `Flip` and `Swap` augmentations during the test time (*i.e*. during navigation) then aggregate pose-predictions. The aggregation consists of two steps: first we transform egomotion estimates for transformed input pairs back to original coordinate system, $\texttt{Flip}^{-1}(\Delta_{\text{pose}}) = (-\Delta x, \Delta y, \Delta z, -\Delta\theta)$, $\texttt{Swap}^{-1}(\Delta_{\text{pose}}) = -\Delta_{\text{pose}}$, and then take the average (illustrated in Fig. 3).

	Dataset	VO		Embedding		Train time		Test time		Navigation metrics ($\times 10^2$)			
	size(M)	Encoder	Size(M)	1FC	2FC	Flip	Swap	Flip	Swap	Success	SPL	SoftSPL	d_G
1	0.5	ResNet18	4.2							64.0±0.9	51.9±1.0	72.6±0.3	57.7±2.5
2	0.5	ResNet18	4.2	✓						70.6±1.9	56.5±1.7	73.3±0.3	43.7±1.7
3	0.5	ResNet18	4.2	✓	✓					72.1±1.0	57.5±0.4	73.4±0.5	42.7±2.6
4	0.5	ResNet18	4.2	✓	✓		✓			69.5±1.3	55.6±1.1	72.5±0.2	50.6±1.7
5	0.5	ResNet18	4.2	✓	✓		✓		✓	71.0±1.3	57.0±1.2	72.5±0.5	53.6±2.4
6	0.5	ResNet18	4.2	✓	✓	✓				73.5±1.8	58.7±1.3	74.0±0.1	39.7±1.9
7	0.5	ResNet18	4.2	✓	✓	✓		✓		75.7±0.8	60.7±0.8	74.1±0.2	37.1±2.3
8	0.5	ResNet18	4.2	✓	✓	✓	✓			73.7±0.3	58.8±0.6	72.8±0.2	45.2±2.9
9	0.5	ResNet18	4.2	✓	✓	✓	✓		✓	75.5±1.2	60.3±0.9	73.5±0.2	40.3±1.9
10	0.5	ResNet18	4.2	✓	✓	✓	✓	✓		76.2±0.9	60.6±0.8	73.3±0.3	39.2±2.0
11	0.5	ResNet18	4.2	✓	✓	✓	✓	✓	✓	77.0±0.8	61.5±0.5	73.9±0.3	37.2±1.0
12	1.5	ResNet18	4.2	✓	✓					77.0±1.3	62.0±1.0	74.3±0.3	38.0±0.9
13	1.5	ResNet18	4.2	✓	✓	✓	✓			80.0±0.9	64.1±0.9	73.8±0.4	37.8±0.7
14	1.5	ResNet18	4.2	✓	✓	✓	✓	✓	✓	85.2±0.5	68.4±0.2	74.9±0.4	31.5±1.5
15	1.5	ResNet50	7.6	✓	✓	✓	✓	✓	✓	88.0±0.6	70.6±0.2	75.5±0.2	26.8±1.7
16	5	ResNet50	7.6	✓	✓	✓	✓	✓	✓	96.0±0.5	76.6±0.4	76.4±0.3	20.1±0.8
17	Ground-Truth Odometry									99.8±0.1	79.8±0.2	77.0±0.2	16.2±1.1

Table 1. **Evaluation on the Gibson v2 4+ validation split.** Results are reported as an average of four evaluations with different seeds. Navigation metrics Success, SPL, SoftSPL, d_G (distance to goal) are subject to $\times 10^2$ multiplication. Tick in a column indicate whether a particular option is turned on. For instance, in row 6 ResNet18 is the VO encoder, action embedding is concatenated to 1-st fully connected and 2-nd fully connected layer, flip augmentation is turned on during training, and navigation metrics are reported with no augmentations during evaluation.

4.4. Training Details

We train the visual odometry model decoupled from the navigation policy – on a static dataset $\mathcal{D} = \{(O_{t-1}, O_t, a_{t-1}, \Delta_{\text{pose}})\}$. This dataset is created by using the oracle to unroll trajectories from which the pairs of RGB-D frames with meta-information about actions taken and egomotions are uniformly sampled (similar dataset collection protocol were used by [9, 39]). We use Gibson 4+ scenes (and Gibson-v2 PointGoal navigation episodes) to generate the VO dataset. We collect the training dataset by uniformly sampling 20% of pairs of observations from train scenes (500k to 5M total training examples) and the validation dataset by sampling 75% of pairs of observations from validation scenes (34k total).

The model is trained with batch size 32, Adam optimizer with learning rate 10^{-4} and mean squared error (MSE) loss for both translation and rotation.

4.5. Dataset and Model Size

We vary the dataset that the VO module is trained on from 500k to 5M observation tuples and replace ResNet18 encoder with ResNet50 encoder.

As the training time increases linearly to the dataset size, we also implemented the distributed VO training pipeline that allows for multi node multi GPU scaling and significantly reduces experiment time. Training on 8 nodes (with 8 GPUs each) runs 6.4 × faster that training on 1 node.

5. Experiments

We report experiments results in Tab. 1. Experiments in rows 1-15 were run for 50 epochs and 90 epochs for row 16 - best HC 2021 PointNav agent. We perform early-stopping via validation loss. To study the impact of different visual odometry modules we fixed the navigation policy (used the same network weights) across all experiments.

5.1. Ablations

In this section we study the importance of proposed additions over a baseline VO model: incorporating meta-information available to the agent by adding action embeddings, `Flip` and `Swap`, and larger datasets. We start from a baseline ResNet18 model (Tab. 1, row 1).

Action embedding. We analyze two possible ways of incorporating meta-information: concatenating the embedding to the *first* FC layer that goes after encoder (Tab. 1, row 2) and concatenating the embedding to *all* FC layers (row 3). Concatenating action embedding to first FC layer improves performance by +7 Success/+5 SPL compared to a baseline (row 2 vs row 1). Concatenating action em-

bedding to all FC layers improves performance further by +1 Success/+1 SPL (row 3 vs row 2). We believe this allows the FC layers to receive more context to learn more accurate egomotion for each action type using shared encoder.

Train-time augmentations. Enriching the VO dataset diversity by applying `Flip` improves performance by +2 Success/+1 SPL (row 6 vs row 3). Interestingly we found `Swap` hurts performance by -2 Success/-2 SPL (row 4 vs row 3) while `Flip` +`Swap` achieves performance equivalent to `Flip` (row 8 vs row 6).

Test-time augmentations. The biggest performance boost from augmentations comes when they are also applied at test-time (navigation). At inference `Swap` and `Flip` are applied (turned no/off) independently. Turning `Flip` on at test-time improves performance by +2 Success/+2 SPL compared to a model with `Flip` on only at train-time (row 10 vs row 6). With `Swap` on at train- and test-time, performance is still worse than achieved by model without `Swap`, -1 Success/-1 SPL (row 5 vs row 3). However, when both `Swap` and `Flip` are on at train- and test-time performance improves further by +1 Success/+1 SPL compared to model with `Flip` (row 11 ws row 10). That is a total improvement of +5 Success/+4 SPL compared to a model trained and evaluated without augmentations (row 11 vs row 3).

Larger dataset. To study the impact of large scale training we increased the training dataset size 3× (from 500k to 1.5M training pairs) following the same dataset collection protocol, described in Sec. 4.4). Without augmentations, increasing dataset size 3× improves performance by +5 Success/+4 SPL (row 12 vs row 3) and by +8 Success/+4 SPL(row 14 vs row 11) with augmentations.

We also examine the impact of augmentations with this larger dataset. Surprisingly, we find that they are *more* influential with a larger training dataset. At train-time, `Swap` +`Flip` improve performance by +2 Success/+1 SPL with a small dataset (row 8 vs row 3) while they improve performance by +3 Success/+2 SPL (row 13 vs row 12) with a large dataset. A test-time, `Swap` +`Flip` improve performance by +3 Success/+3 SPL (row 11 vs row 8) with a small dataset while they improve performance by +5 Success/+4 SPL (row 14 vs row 13) with a large dataset.

Deeper encoder. We find that training with more sophisticated encoder architecture (ResNet50 instead of ResNet18) improves navigation performance further by +3 Success/+3 SPL (row 15 vs row 14). Given the additional representational capacity of ResNet50, we further increase the training dataset size to 5M pairs. This improves performance by +8 Success/+6 SPL (row 16 vs row 15).

Dataset transfer. We examine how the two components of our agent transfer from their training dataset, Gibson, to the Matterport3D dataset [5]. We find that while the performance of the agent with ground-truth localization is reduced by only a small amount, -6 Succes/-6 SPL (Tab. 2, row 5 vs row 2), the performance of the agent with visual odometry is reduced by considerably more, -19 Success/-18 SPL (row 6 vs row 3). This is inline with observed in the idealized case where Depth-only agents (like our agent with ground-truth) transfer from Gibson to Matterport3D well, agents with RGB-D (like our agent with VO) transfer poorly [24, 25]. This leaves the question – is there a universal (cross-dataset) VO module? We anticipate creating one will require training on multiple large-scale datasets.

	Dataset	Policy	Navigation metrics (×10²)			
			Success	SPL	SoftSPL	d_G
1	Gibson	Oracle	98.6±0.1	84.5±0.1	80.5±0.2	30.6±1.5
2	Gibson	Learned+GT	99.8±0.1	79.8±0.2	77.0±0.2	16.2±1.1
3	Gibson	Learned+VO	96.0±0.5	76.6±0.4	76.4±0.3	20.1±0.8
4	MP3D	Oracle	98.7±0.4	85.4±0.4	83.2±0.2	34.3±0.9
5	MP3D	Learned+GT	94.4±0.6	71.8±0.7	70.7±0.6	123.5±8.7
6	MP3D	Learned+VO	79.4±1.7	60.9±1.3	69.1±0.3	142.9±16.6

Table 2. **Dataset transfer.** We evaluate how well the components of our agent transfer from its training dataset (Gibson v2) to the validation dataset of Matterport3D v2. We find that while the policy transfers well, visual odometry performance suffers.

Rank	Participant team	Navigation metrics (×10²)			
		Success	SPL	SoftSPL	d_G
1	VO for Realistic PointGoal (Ours)	**94**	**74**	**76**	**21**
2	inspir.ai robotics	91	70	71	70
3	VO2021 (Zhao *et al*. [39])	78	59	69	53
4	Differentiable SLAM-net ([15])	65	47	60	174

Table 3. **Habitat Challenge 2021** benchmark test-standard split (retrieved 2021-Nov-16). The work of 'inspir.ai robotics' is concurrent unpublished work.

5.2. Habitat Challenge 2021 PointNav Track

We evaluate our most performant agent (Tab. 1, row 16) on the Habitat Challenge 2021 benchmark test-std split. Our agent achieves 94% Success and 74% SPL (Tab. 3) on the test-std split. This is an increase of +16% Success/+15% SPL over prior published state-of-the-art, Zhao *et al*. [39]. An unpublished concurrent work increased performance to 91% Success/70% SPL and our method improves upon that further.

While our results do not effectively 'solve' PointGoal navigation under realistic settings, they improve performance significantly and add more evidence that navigation without building an explicit map *is* possible, even under harsh realistic conditions.

6. Real-World Transfer

We perform an initial exploration of our method in reality and deploy our learned agent on a LoCoBot with no sim2real adaptation. Across 9 episodes, it achieves 11% Success, 71% SoftSPL, and makes it 92% of the way to the goal (SoftSuccess). Based on the navigation videos

provided on the website [5] the agent does a good job avoiding obstacles. These initial results show promise, and adaptation methods may improve the performance.

7. Related Work

Autonomous navigation has long been a subject of research in robotics and computer vision [10,18,20]. With advances in computer vision and deep learning, there has been a renewed interest in the use of learning to derive navigation policies for a variety of tasks (such as rearrangement [3,29], visual navigation, [1,4,7], and vision-and-language [1,16]).

Classical vs learned navigators. Classical approaches decompose the problem into a sequence of sub-tasks, such as localization, mapping, planning, and control. Each of the sub-tasks is addressed separately and corresponding solutions are then composed into one pipeline. When properly tuned, such methods can perform well. Wijmans *et al*. [33] showed that learned approaches can outperform their classical counterparts with sufficient data and training.

Visual odometry for navigation. Given the importance of localization for navigation, design choices of the CNN-based relative pose regression given the two consecutive RGB/RGB-D frames and their influence on the downstream navigation metrics of navigation agents has been a subject of prior works [6,9,15,23,39].

Neural SLAM [6] integrates learning into classical modular SLAM components and estimates agent pose change by using its predicted egocentric map to update the noisy localization sensor on a LoCoBot. Built on top of Neural SLAM architecture, Occupancy Anticipation [23] learns to estimate egomotion directly from RGB-D input and uses egocentric occupancy maps as an auxiliary signal. Differentiable SLAM-net [15] jointly optimizes all SLAM components by backpropagating through a particle-filter based SLAM algorithm. Such approach significantly improved environment map accuracy that translated into improved downstream navigation performance.

Approaches that do not built an explicit map divide learning agent dynamics and visual odometry (VO) into two separate components. Initial attempts achieved worse results than approaches that use an explicit map [9]. Zhao *et al*. [39] focused on improving VO for navigation and showed that map-less approaches can outperform map-building approaches. We continue improvements to VO for navigation and reduce the gap between state-of-the-art performance and an oracle from 31% SPL to 7% SPL.

8. Concluding Remarks

We studied the question *'can an autonomous agent navigate in a new environment without building an explicit map?'* under harsh realistic conditions. Towards answering this question we first demonstrated that when given ground-truth localization (`GPS+Compass`) map-less agents are able to overcome actuation noise and sensor noise and learn to navigate with near-perfect performance, thereby identifying localization as the limiting factor.

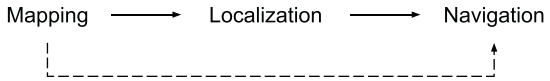

Figure 5. We study the direct link between mapping and navigation. We find additional evidence that this link is weak. We do not study indirect links however. Indirect links, like the link from mapping to localization to navigation may still be strong.

To improve localization performance, we presented a series of broadly-applicable additions to visual odometry (VO) that improve performance from 64% Success/52% SPL to 96% Success/77% SPL. While our results do not effectively 'solve' PointGoal navigation in the realistic setting, they improve performance significantly and add more evidence that navigation without building an explicit map *is* possible even under harsh realistic conditions.

Limitations. While our work presents a significant advance in map-less navigation methods for realistic conditions it has several limitations. 1) Embodiment specificity. While our VO model and training procedure are policy agnostic, they are not embodiment agnostic. The importance of action embeddings implies that relaxing this will be challenging, meaning that the VO model may need to be re-trained for each embodiment, which is wasteful. 2) Dataset specificity. Similarly, our learned VO model does not transfer well between datasets and may need to be re-trained for each dataset. We believe large-scale multi-dataset training may be a solution but this remains an open question. 3) Compute requirements. Our best navigation policy used a total of 7.5 billion steps of experience. Training our best VO model required first generating 5M training pairs and then training on 64 GPUs (∼5,000 GPU hours total). High compute requirements were swiftly reduced for PointNav-v1 [26,32,36] and we anticipate they will reduce for PointNav-v2 too, but this remains an open direction.

With regard to the core question, we studied the direct link between mapping and navigation and found increasing evidence that it is a weak link. We have not studied indirect links between mapping and navigation and these may be strong. For instance, there is reason to believe that mapping is needed for accurate localization over long time horizons and localization is needed for navigation (illustrated in Fig. 5). Studying indirect links is an avenue for future work.

Acknowledgements. The Georgia Tech effort was supported in part by NSF, ONR YIP, and ARO PECASE. EW is supported in part by an ARCS fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

[5] https://rpartsey.github.io/pointgoalnav

References

[1] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 1, 2, 8

[2] Nicholas Ayache and Olivier D Faugeras. Building, registrating, and fusing noisy visual maps. *The International Journal of Robotics Research*, 7(6):45–65, 1988. 1

[3] Dhruv Batra, Angel X. Chang, Sonia Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020. 8

[4] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*, 2020. 8

[5] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. 7

[6] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *International Conference on Learning Representations*, 2020. 8

[7] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. Soundspaces: Audio-visual navigation in 3d environments. In *ECCV*, 2020. 8

[8] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5556–5565, 2015. 3

[9] Samyak Datta, Oleksandr Maksymets, Judy Hoffman, Stefan Lee, Dhruv Batra, and Devi Parikh. Integrating egocentric localization for more realistic point-goal navigation agents. *arXiv: Computer Vision and Pattern Recognition*, 2020. 2, 3, 6, 8

[10] H. Durrant-Whyte, D. Rye, and E. Nebot. Localization of autonomous guided vehicles. In *Robotics Research*, 1996. 8

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 3, 5

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 3

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, volume 1, pages 448–456, 2015. 5

[14] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Are we making real progress in simulated environments? measuring the sim2real gap in embodied visual navigation. *arXiv: Computer Vision and Pattern Recognition*, 2019. 3

[15] Péter Karkus, Shaojun Cai, and David Hsu. Differentiable slam-net: Learning particle slam for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2815–2825, 2021. 7, 8

[16] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *European Conference on Computer Vision*, pages 104–120, 2020. 8

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of The ACM*, 60(6):84–90, 2017. 2

[18] Hans Moravec. Locomotion, vision and intelligence. In Michael Brady and Richard Paul, editors, *Proceedings of Robotics Research - The First International Symposium*, pages 215–224. MIT Press, August 1984. 8

[19] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv: Robotics*, 2019. 3, 5

[20] N Nilsson. Shakey the robot, 1984. 1, 8

[21] John O'keefe and Lynn Nadel. *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978. 1

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5

[23] Santhosh K. Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. In *ECCV (5)*, pages 400–418, 2020. 8

[24] Santhosh K. Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *arXiv preprint arXiv:2109.08238*, 2021. 1, 2, 3, 7

[25] Manolis Savva, Jitendra Malik, Devi Parikh, Dhruv Batra, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, and Vladlen Koltun. Habitat: A platform for embodied ai research. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9339–9347, 2019. 3, 7

[26] Brennan Shacklett, Erik Wijmans, Aleksei Petrenko, Manolis Savva, Dhruv Batra, Vladlen Koltun, and Kayvon Fatahalian. Large batch simulation for deep reinforcement learning. *Int. Conf. Learn. Represent.*, 2021. 8

[27] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer, 1990. 1

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 5

[29] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021. 8

[30] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics (intelligent robotics and autonomous agents), 2005. 1

[31] Edward C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948. 1

[32] Erik Wijmans, Irfan Essa, and Dhruv Batra. How to train pointgoal navigation agents on a (sample and compute) budget. *arXiv preprint arXiv:2012.06117*, 2020. 8

[33] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *Eighth International Conference on Learning Representations*, 2020. 2, 3, 4, 8

[34] Yuxin Wu and Kaiming He. Group normalization. *arXiv: Computer Vision and Pattern Recognition*, 2018. 5

[35] Fei Xia, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. 2, 3

[36] Joel Ye, Dhruv Batra, Erik Wijmans, and Abhishek Das. Auxiliary tasks speed up learning pointgoal navigation. *Conference on Robot Learning (CoRL)*, 2020. 8

[37] Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6022–6031, 2019. 2

[38] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2017. 2

[39] Xiaoming Zhao, Harsh Agrawal, Dhruv Batra, and Alexander G. Schwing. The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16127–16136, 2021. 1, 2, 3, 5, 6, 7, 8