ROBUSTNAV: Towards Benchmarking Robustness in Embodied Navigation

1. Supplementary Material

Contents

1.1. Overview

This supplementary document is organized as follows. In Sec. 1.2, we describe in detail the task specifications for POINTNAV and OBJECTNAV. In Sec. 1.3, we provide details about the architecture adopted for POINTNAV and OBJECTNAV agents and how they are trained. In Sec. 1.4, we include more plots demonstrating the kinds of behaviors POINTNAV and OBJECTNAV agents exhibit under corruptions (`RGB-D` variants in addition to the `RGB` variants in Sec. 5.2 of the main paper). In Sec. 1.5, we provide more results demonstrating degradation in performance at severity set to 3 (for `vis` corruptions with controllable severity levels; excluded from the main paper due to space constraints) and break down performance degradation by episode difficulty. We also report degradation in task performance of POINTNAV agents in the presence of compositions of multiple `vis` corruptions and depth noise models. In Sec. 1.6, we provide more details about the `vis` and `dyn` corruptions present in ROBUSTNAV.

1.2. Task Specifications

We describe the task-specifications (as outlined in Sec. 3 of the main paper) for the ones included in ROBUSTNAV in detail. Note that while ROBUSTNAV currently supports navigation heavy tasks, the corruptions included can easily be extended to other embodied tasks (and associated simulators) that share the same modalities, for instance, tasks involving vision and language guided navigation or having interaction components.

POINTNAV. In POINTNAV, an agent is spawned at a random location and orientation in an environment and asked to navigate to goal coordinates specified relative to the agent's position. This is equivalent to the agent being equipped with a GPS+Compass sensor (providing relative location and orientation with respect to the agent's current position). Note that the agent does not have access to any "map" of the environment and must navigate based solely on sensory inputs from a visual `RGB` (or `RGB-D`) and GPS+Compass sensor. An episode is declared successful if the POINTNAV agent stops (by "intentionally" invoking an **end** action) within 0.2m of the goal location.

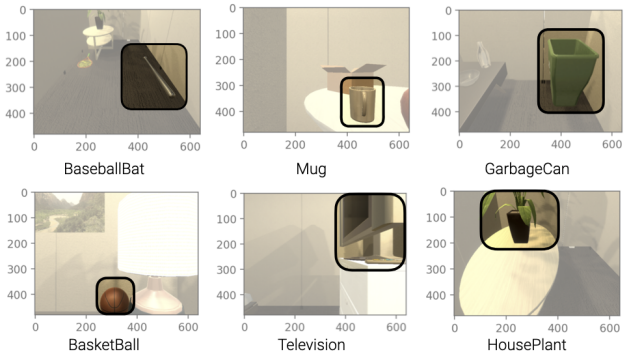

Figure 1. **OBJECTNAV Target Objects.** We present a few examples of the target objects considered for OBJECTNAV agents in ROBOTHOR as viewed from the agent's ego-centric `RGB` frame under successful episode termination conditions.

OBJECTNAV. In OBJECTNAV, an agent is spawned at a random location and orientation in an environment as is asked to navigate to a specified "object" category (e.g, *Television*) that exists in the environment. Unlike POINTNAV, an OBJECTNAV agent does not have access to a GPS+Compass sensor and must navigate based solely on the specified target and visual sensor inputs – `RGB` (or `RGB-D`). An episode is declared successful if the OBJECTNAV agent (1) stops (by "intentionally" invoking an **end** action) within 1.0m of the target object and (2) has the target object within it's ego-centric view. We consider 12 object categories present in the ROBOTHOR scenes for our OBJECTNAV experiments. These are AlarmClock, Apple, BaseballBat, BasketBall, Bowl, GarbageCan, HousePlant, Laptop, Mug, SprayBottle, Television and Vase (see Fig. 1 for a few examples in the agent's ego-centric frame).

Calibration Budget. Our calibration budget ($\sim$ 166k steps) was decided based on how long it takes an agent to reasonably recover "lost" performance via supervised fine-

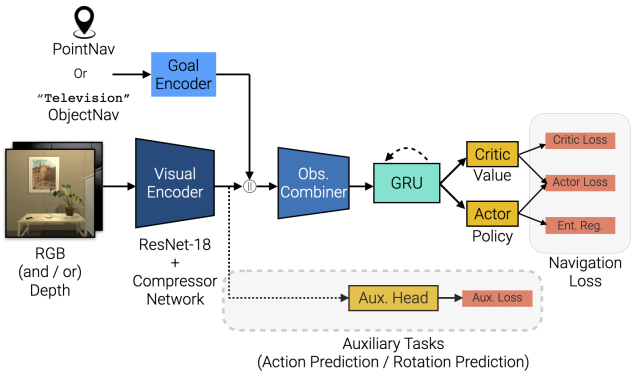

Figure 2. **Agent Architecture.** We show the general architecture adopted for our POINTNAV and OBJECTNAV agents – convolutional units to encode observations followed by recurrent policy networks. The auxiliary task heads are used when we consider pre-training or adaptation using PAD [8].

tuning in the corrupt environment. This corresponds to $< 0.3\%$ of the training time for both tasks. In terms of real-world agent interaction time, following LoCoBot specifications, the "calibration budget" corresponds to an upper bound of ~ 2.40 days. We intended to start with a higher value for the budget, since even supervised fine-tuning takes significantly long (that too in simulation) to recover lost performance. As navigation agents progressively become better and robust, this can be reduced.

1.3. Navigation Agents

We highlight describe the architecture of the agents studied in ROBUSTNAV and provide additional training details.

Base Architecture. We consider standard neural architectures (akin to [19, 18]) for both POINTNAV and OBJECTNAV– convolutional units to encode observations followed by recurrent policy networks to predict action distributions. Concretely, our agent architecture consists of four major components – a visual encoder, a goal encoder, a target observation combiner and a policy network (see Fig. 2). The visual encoder (for `RGB` and `RGB-D`) agents consists of a frozen ResNet-18 [9] encoder (till the last residual block) pretrained on ImageNet [6], followed by a learnable compressor network consisting of two convolutional layers of kernel size 1, each followed by ReLU activations ($512 \times 7 \times 7 \rightarrow 128 \times 7 \times 7 \rightarrow 32 \times 7 \times 7$). The goal encoder encodes the specified target – a goal location in polar coordinates (r, θ) for POINTNAV and the target object token (*e.g., Television*) for OBJECTNAV. For POINTNAV, the goal is encoded via a linear layer (2×32). For OBJECTNAV, the goal is encoded via an embedding layer (12×32) set to encode one of the 12 object categories. The goal embedding and output of the visual encoder are then concatenated and further passed through the target observation combiner network consisting of two convolutional layers of kernel size 1 ($64 \times 7 \times 7 \rightarrow 128 \times 7 \times 7 \rightarrow 32 \times 7 \times 7$). The output of the target observation combiner is flattened and then fed to the policy network – specifically, to a single layer GRU (hidden size 512), followed by linear actor and critic heads used to predict action distributions and value estimates.

Auxiliary Task Heads. In Sec. 5.3 of the main paper, we investigate if self-supervised approaches, particularly, Policy Adaptation during Deployment (PAD) [8] help in resisting performance degradation due to `vis` corruptions. Incorporating PAD involves training the vanilla agent architectures (as highlighted before) with self-supervised tasks (for pre-training as well as adaptation in a corrupt target environment) – namely, Action Prediction (AP) and Rotation Prediction (RP). In Action-Prediction (AP), given two successive observations in a trajectory, an auxiliary head is tasked with predicting the intermediate action and in Rotation-Prediction (RP), the input observation is rotated by $0°$, $90°$, $180°$, or $270°$ uniformly at random before feeding to the agent and an auxiliary head is asked to to predict the rotation bin. For both AP and RP, the auxiliary task heads operate on the encoded visual observation (as shown in Fig. 2). To gather samples in the target environment (corrupt or otherwise), we use data collected from trajectories under the source (clean) pre-trained policy – *i.e.,* the visual encoder is updated online as observations are encountered under the pre-trained policy.

Training and Evaluation Details. As mentioned earlier, we train our agents with DD-PPO [19] (a decentralized, distributed version of the Proximal Policy Optimization Algorithm [17]) with Generalized Advantage Estimation [16]. We use rollout lengths $T = 128$, 4 epochs of PPO with 1 mini-batch per-epoch. We set the discount factor to $\gamma = 0.99$, GAE factor to $\tau = 0.95$, PPO clip parameter to 0.1, value loss coefficient to 0.5 and clip the gradient norms at 0.5. We use the the Adam optimizer [12] with a learning rate of $3e-4$ with linear decay. The reward structure used is as follows – if $\texttt{R} = 10.0$ denotes the terminal reward obtained at the end of a "successful" episode and $\lambda = -0.01$ denotes a slack penalty to encourage efficiency, then the reward received by the agent at time-step t can be expressed as,

$$r_t = \underbrace{\texttt{R} \cdot \mathbb{I}_{\text{success}}}_{\text{success reward}} - \underbrace{\Delta_t^{\text{Geo}}}_{\text{reward shaping}} + \underbrace{\lambda}_{\text{slack reward}}$$

where Δ_t^{Geo} is the change in geodesic distance to the goal and $\mathbb{I}_{\text{Success}}$ indicates where the episode was successful (1) or not (0). During evaluation, we allow an agent to execute a maximum of 300 steps – if an agent doesn't call **end** within 300 steps, we forcefully terminate the episode. All agents are trained under LocoBot calibrated actuation noise models from [5] – $\mathcal{N}(0.25\text{m}, 0.005\text{m})$ for translation and $\mathcal{N}(30°, 0.5°)$ for rotation. During evaluation, with the exception of circumstances when Motion Bias (S) is present, we use the same actuation noise models (in addition to dyn corruptions when applicable). We train our POINTNAV agents for $\sim$ 75M steps and OBJECTNAV agents for

$\sim$ 300M steps (both RGB and RGB-D variants).

1.4. Behavior Analysis

In Sec. 5.2 of the main paper, we try to understand the idiosyncrasies exhibited by the navigation agents under corruptions. Specifically, we look at the number of collisions as observed through the number of failed actions in RoboTHOR, the closest the agent arrives to the target in an episode and Stop-Fail (Pos) and Stop-Fail (Neg). Since for both PointNav and ObjectNav, success depends on a notion of "intentionality" [2] – the agent calls an **end** action when it believes it has reached the goal – we use both Stop-Fail (Pos) and Stop-Fail (Neg) to assess how corruptions impact this "stopping" mechanism of the agents. Stop-Fail (Pos) measures the fraction of times the agent calls an **end** action when the goal is not in range[1], out of the number of times the agent calls an **end** action. Stop-Fail (Neg) measures the fraction of times the agent fails to invoke an **end** action when the goal is in range, out of the number of steps the goal is in range in an episode. Both are averaged across evaluation episodes. In addition to the above aspects, we also measure the average distance to the goal at episode termination. Here we report these measures for PointNav and ObjectNav agents trained with RGB and RGB-D sensors in Fig. 3 (RGB-D variants in addition to the RGB agents in Fig.4 of the main paper).

We find that across RGB and RGB-D variants, (1) agents tend to collide more often under corruptions (Fig. 3, col 1), (2) agents generally end up farther from the target at episode termination under corruptions (Fig. 3, col 2) and (3) agents tend to be farther from the target under corruptions even in terms of minimum distance over an episode (Fig. 3, col 3). We further note that the effect of corruptions on the agent's stopping mechanism is more pronounced for ObjectNav as opposed to PointNav (Fig. 3, cols 4 & 5).

To further understand the extent to which a worse stopping mechanism impacts the agent's performance, in Fig. 4, we compare the agents' success rate (SR) with a setting where the agent is equipped with an oracle stopping mechanism (forcefully call **end** when goal is in range). For both ObjectNav RGB and RGB-D, we find that the presence of vis and vis+dyn corruptions affects success significantly compared to the clean settings (Fig. 4, black bars).

1.5. Degradation Results

Habitat Challenge Results. We further investigate the degree to which more sophisticated PointNav agents, composed of map-based architectures, are susceptible to vis corruptions. Specifically, we evaluate the performance of the winning entry of Habitat Challenge (HC) 2020 [1] – Occupancy Anticipation [15] on the Gibson [20] validation scenes (see Table. 1). We evaluate the performance of OccAnt (for RGB and RGB-D; based on provided checkpoints) when vis corruptions are introduced (1) over clean settings under noise-free conditions (rows 1-3 in Table. 1) and (2) by replacing the RGB noise under Habitat Challenge (HC) conditions (rows 4-6 in Table. 1). Under noise free conditions, we note that degradation in performance from the clean settings is more pronounced for the RGB agents as opposed to the RGB-D variants. Under HC conditions, we note that the RGB-D variants suffer significant degradation in per-

[1]The goal in range criterion for PointNav checks whether the target is within the threshold distance. For ObjectNav, this includes a visibility criterion in addition to taking distance into account.

	SR ↑	SPL ↑	Len.	SR ↑	SPL ↑	Len.
Noise Free	RGB			RGBD		
(1) Clean	88.90	70.70	240.897	92.50	78.20	185.255
(2) Low-Light.	63.30	32.90	566.286	91.90	75.10	207.994
(3) Spatter	47.50	18.40	728.074	94.80	78.50	181.732
HC Conditions	RGB			RGBD		
(4) HC RGB Noise	N/A	N/A	N/A	65.90	49.50	104.167
(5) Low-Light.	N/A	N/A	N/A	60.50	45.50	107.932
(6) Spatter	N/A	N/A	N/A	41.60	32.10	110.630

Table 1. **OccAnt [15] results on Gibson [20] val.** Rows 1-3 are when vis corruptions are introduced over clean settings under noise-free conditions, based on the checkpoint used to report results in the publication. Rows 4-6 are when RGB noise under Habitat Challenge (HC) conditions is replaced with the vis corruptions, based on the HC submission checkpoint. Len indicates episode length. N/A implies checkpoint not available. Severity for Low-Lighting and Spatter is set to 5 (worst).

	SR ↑	SPL ↑	SR ↑	SPL ↑
Corruptions	Clean Depth		Noisy Depth	
(1) Clean	98.54	84.60	85.26	59.67
(2) Low Lighting	99.45	84.97	77.89	54.26
(3) Speckle Noise	98.73	84.66	75.07	50.93

Table 2. **Depth Corruptions.** Degradation in task performance of pretrained PointNav RGB-D (trained for $\sim$ 75M frames) agents when evaluated in the presence of noisy depth in addition to corrupt RGB observations. Depth noise is based on the redwood depth noise model from [4] (designed for PrimeSense sensors). Severity for Low Lighting and Speckle Noise is set to 5 (worst).

PointNav	RGB		RGB-D	
Corruptions	SR ↑	SPL ↑	SR ↑	SPL ↑
(1) Clean	98.82	83.13	98.54	84.60
(2) Low Lighting	94.36	75.15	99.45	84.97
(3) Motion Blur	95.72	73.37	99.36	85.36
(4) Camera Crack (CC)	82.07	63.83	95.72	81.21
(5) CC + Low Lighting	70.16	51.78	98.18	83.33
(6) CC + Motion Blur	26.02	18.42	97.27	82.45

Table 3. **Corruption Compositions.** Degradation in task performance of pretrained PointNav (trained for $\sim$ 75M frames) in the presence of composition of viscorruptions (rows 5 and 6). Severity of Low Lighting and Motion Blur is set to 5 (worst).

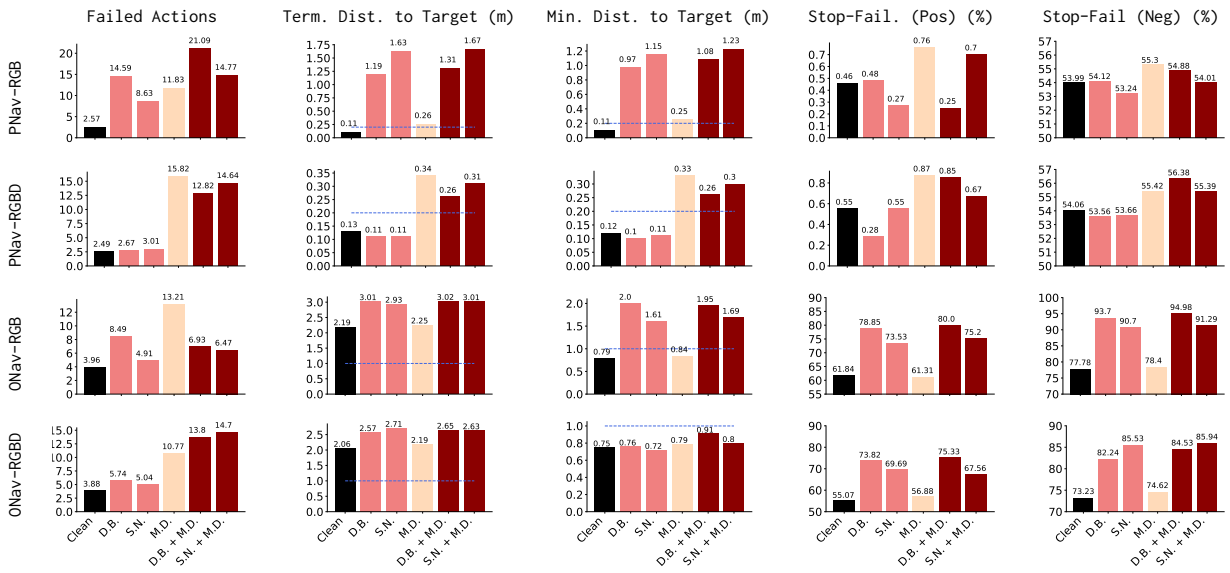

Figure 3. **Agent Behavior Analysis.** To understand agent behaviors, we report the breakdown of four metrics: Number of collisions as observed through `Failed Action` (*first column*), distance to target at episode termination as measured by `Term. Dist. to Target` (*second column)*, closest agent was to target as measured by `Min. Dist. to Target` (*third column*), and failure to appropriately end an episode either when out of range – `Stop-Fail (Pos)` (*fourth column*), or in range – `Stop-Fail (Neg)` (*fifth column*). Each behavior is reported for both PointNav (RGB-*first row*, RGBD-*second row*) and ObjectNav (RGB-*third row*, RGBD-*fourth row*) within a clean and five corrupt settings: Defocus Blur (D.B.), Speckle Noise (S.N.), Motion Drift (M.D.), Defocus Blur + Motion Drift, and Speckle Noise + Motion Drift. ■ is clean, ■ is `vis` corruptions, ■ is `dyn` corruptions and ■ is `vis+dyn` corruptions. Blue lines in column 2 and 3 indicate the distance threshold for goal in range. Severities for S.N. and D.B. are set to 5 (worst).

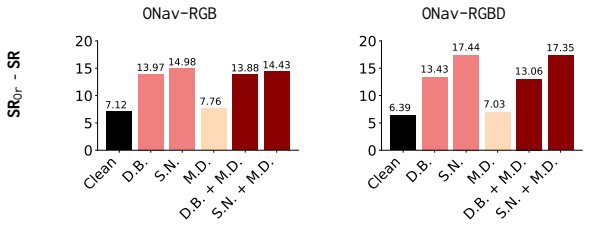

Figure 4. **Effect of Degraded Stopping Mechanism.** To understand the extent to which a degraded stopping mechanism under corruptions affects ObjectNav RGB agent performance, we look the difference between the agent's success rate (SR) compared to the setting where the agent is equipped with an oracle stopping mechanism. SR_{Or} denotes success rate when an **end** action is forcefully called in an episode whenever the goal is in range. We consider one clean and five corrupt settings: Defocus Blur (D.B.), Speckle Noise (S.N.), Motion Drift (M.D.), Defocus Blur + Motion Drift, and Speckle Noise + Motion Drift. ■ is clean, ■ is `vis` corruptions, ■ is `dyn` corruptions and ■ is `vis+dyn` corruptions. Severities for S.N. and D.B. are set to 5 (worst).

formance when RGB noise is replaced with `vis` corruptions.

Depth Corruptions. We believe modeling corruptions in depth is not as trivial as modeling corruptions in the RGB observations. For instance, it's unclear how motion blur (due to fast, jittery movements) affects depth observations. Given this, and the fact that we intend to factorize the contributions of the depth and RGB sensors under corruptions, we decided to solely rely on RGB corruptions as a starting step. More sophisticated depth corruptions – such as Redwood, SimKinect, etc. – will be incorporated in future versions of RobustNav. As an anecdotal point of reference, we report the performance of trained PointNav RGB-D agents in presence of the redwood depth noise model [4] (for PrimeSense derived sensors), in Table. 2.

Corruption Compositions. RobustNav is not restricted to having at most one `vis`, and / or one `dyn` corruption(s) in the target environment – it can support a composition of multiple corruptions as well. In Table. 3, we report the performance of PointNav agents under a subset of such `vis` compositions. We restrict ourselves to individual corruptions in the main paper to factorize the effect of each corruption on navigation performance.

More Degradation Results. In Table. 4, we report the degradation in performance (relative to clean settings) of PointNav and ObjectNav agents when operating under `vis`, `dyn` and `vis+dyn` corruptions. We report mean and standard error values across 3 evaluation runs under actuation noise (wherever applicable). For `vis` corruptions with controllable severity levels – Motion Blur, Low-Lighting, Defocus Blur, Speckle Noise and Spatter – we report results with severities set to 3 and 5 (identified by S3 and S5; excluded from the main paper due to space constraints) – for both `vis` and `vis+dyn` settings. We note that unlike the RGB-D variants, for PointNav RGB agents, performance drops more as severity levels increase (increasing degrada-

				PointNav				ObjectNav			
				RGB		RGB-D		RGB		RGB-D	
#	Corruption ↓	V	D	SR ↑	SPL ↑	SR ↑	SPL ↑	SR ↑	SPL ↑	SR ↑	SPL ↑
1	Clean			$98.97_{\pm 0.18}$	$83.45_{\pm 0.27}$	$99.24_{\pm 0.15}$	$85.00_{\pm 0.25}$	$31.78_{\pm 0.81}$	$14.50_{\pm 0.47}$	$35.43_{\pm 0.83}$	$17.57_{\pm 0.52}$
2	Low Lighting (S3)	✓		$97.45_{\pm 0.27}$	$80.53_{\pm 0.33}$	$99.09_{\pm 0.17}$	$84.91_{\pm 0.26}$	$21.55_{\pm 0.72}$	$8.91_{\pm 0.38}$	$27.55_{\pm 0.78}$	$13.08_{\pm 0.47}$
3	Low Lighting (S5)	✓		$93.42_{\pm 0.43}$	$74.88_{\pm 0.43}$	$99.27_{\pm 0.15}$	$85.04_{\pm 0.25}$	$11.69_{\pm 0.56}$	$4.90_{\pm 0.30}$	$23.26_{\pm 0.74}$	$10.61_{\pm 0.43}$
4	Motion Blur (S3)	✓		$98.82_{\pm 0.19}$	$80.64_{\pm 0.29}$	$98.91_{\pm 0.18}$	$84.62_{\pm 0.26}$	$18.57_{\pm 0.68}$	$8.18_{\pm 0.37}$	$24.32_{\pm 0.75}$	$11.52_{\pm 0.44}$
5	Motion Blur (S5)	✓		$96.15_{\pm 0.34}$	$73.24_{\pm 0.38}$	$99.06_{\pm 0.17}$	$85.01_{\pm 0.25}$	$10.50_{\pm 0.93}$	$4.71_{\pm 0.51}$	$18.26_{\pm 0.83}$	$7.87_{\pm 0.45}$
6	Camera Crack	✓		$81.56_{\pm 0.68}$	$63.48_{\pm 0.59}$	$96.00_{\pm 0.34}$	$81.48_{\pm 0.36}$	$7.06_{\pm 0.45}$	$3.54_{\pm 0.28}$	$27.28_{\pm 0.78}$	$13.49_{\pm 0.48}$
7	Defocus Blur (S3)	✓		$94.63_{\pm 0.39}$	$73.28_{\pm 0.41}$	$98.79_{\pm 0.19}$	$84.47_{\pm 0.26}$	$15.59_{\pm 0.63}$	$6.99_{\pm 0.36}$	$22.07_{\pm 0.72}$	$9.75_{\pm 0.41}$
8	Defocus Blur (S5)	✓		$75.83_{\pm 0.75}$	$53.48_{\pm 0.59}$	$99.03_{\pm 0.17}$	$85.44_{\pm 0.25}$	$4.20_{\pm 0.35}$	$1.81_{\pm 0.20}$	$17.66_{\pm 0.67}$	$7.47_{\pm 0.36}$
9	Speckle Noise (S3)	✓		$89.23_{\pm 0.54}$	$68.18_{\pm 0.53}$	$98.85_{\pm 0.19}$	$84.58_{\pm 0.27}$	$14.92_{\pm 0.62}$	$6.54_{\pm 0.34}$	$24.05_{\pm 0.75}$	$10.34_{\pm 0.42}$
10	Speckle Noise (S5)	✓		$66.70_{\pm 0.82}$	$47.92_{\pm 0.67}$	$98.97_{\pm 0.18}$	$84.79_{\pm 0.26}$	$8.68_{\pm 0.49}$	$3.90_{\pm 0.28}$	$17.69_{\pm 0.67}$	$7.42_{\pm 0.36}$
11	Lower-FOV	✓		$43.25_{\pm 0.86}$	$32.39_{\pm 0.68}$	$89.44_{\pm 0.54}$	$73.92_{\pm 0.50}$	$10.17_{\pm 0.53}$	$2.50_{\pm 0.19}$	$10.02_{\pm 0.52}$	$4.89_{\pm 0.31}$
12	Spatter (S3)	✓		$38.40_{\pm 0.85}$	$25.53_{\pm 0.59}$	$98.64_{\pm 0.20}$	$84.00_{\pm 0.28}$	$7.06_{\pm 0.45}$	$3.81_{\pm 0.29}$	$23.20_{\pm 0.74}$	$9.71_{\pm 0.40}$
13	Spatter (S5)	✓		$34.64_{\pm 0.83}$	$25.55_{\pm 0.64}$	$99.30_{\pm 0.14}$	$84.68_{\pm 0.25}$	$7.79_{\pm 0.47}$	$2.71_{\pm 0.22}$	$21.43_{\pm 0.72}$	$9.98_{\pm 0.42}$
14	Motion Bias (S)		✓	$95.69_{\pm 0.35}$	$77.05_{\pm 0.38}$	$96.60_{\pm 0.32}$	$79.22_{\pm 0.35}$	$33.21_{\pm 0.82}$	$14.88_{\pm 0.47}$	$34.98_{\pm 0.83}$	$16.70_{\pm 0.51}$
15	Motion Drift		✓	$95.94_{\pm 0.34}$	$76.32_{\pm 0.35}$	$93.57_{\pm 0.43}$	$75.09_{\pm 0.40}$	$28.55_{\pm 0.79}$	$13.30_{\pm 0.46}$	$34.37_{\pm 0.83}$	$16.43_{\pm 0.50}$
16	Motion Bias (C)		✓	$92.27_{\pm 0.47}$	$77.48_{\pm 0.46}$	$93.11_{\pm 0.44}$	$79.04_{\pm 0.45}$	$30.47_{\pm 0.80}$	$13.20_{\pm 0.45}$	$32.72_{\pm 0.82}$	$15.67_{\pm 0.50}$
17	PyRobot [14] (ILQR) Mul. = 1.0		✓	$95.18_{\pm 0.37}$	$67.45_{\pm 0.37}$	$96.18_{\pm 0.33}$	$69.48_{\pm 0.35}$	$32.54_{\pm 0.82}$	$11.65_{\pm 0.39}$	$36.86_{\pm 0.84}$	$14.24_{\pm 0.44}$
18	Motor Failure		✓	$20.84_{\pm 0.71}$	$17.91_{\pm 0.62}$	$21.41_{\pm 0.71}$	$18.39_{\pm 0.62}$	$4.60_{\pm 0.37}$	$2.88_{\pm 0.26}$	$6.06_{\pm 0.42}$	$3.65_{\pm 0.28}$
19	Defocus Blur (S3) + Motion Bias (S)	✓	✓	$92.72_{\pm 0.45}$	$68.61_{\pm 0.43}$	$97.45_{\pm 0.27}$	$79.70_{\pm 0.32}$	$14.40_{\pm 0.61}$	$6.15_{\pm 0.33}$	$22.40_{\pm 0.73}$	$9.20_{\pm 0.39}$
20	Defocus Blur (S5) + Motion Bias (S)	✓	✓	$75.80_{\pm 0.75}$	$50.76_{\pm 0.58}$	$97.00_{\pm 0.30}$	$79.81_{\pm 0.33}$	$5.66_{\pm 0.40}$	$2.34_{\pm 0.22}$	$17.53_{\pm 0.66}$	$7.07_{\pm 0.35}$
21	Speckle Noise (S3) + Motion Bias (S)	✓	✓	$86.62_{\pm 0.59}$	$63.20_{\pm 0.54}$	$96.85_{\pm 0.30}$	$79.23_{\pm 0.34}$	$14.92_{\pm 0.62}$	$6.31_{\pm 0.34}$	$24.72_{\pm 0.75}$	$10.04_{\pm 0.41}$
22	Speckle Noise (S5) + Motion Bias (S)	✓	✓	$64.36_{\pm 0.83}$	$44.38_{\pm 0.66}$	$96.78_{\pm 0.31}$	$79.49_{\pm 0.34}$	$8.95_{\pm 0.50}$	$3.85_{\pm 0.27}$	$18.39_{\pm 0.68}$	$7.49_{\pm 0.36}$
23	Spatter (S3) + Motion Bias (S)	✓	✓	$37.25_{\pm 0.84}$	$23.83_{\pm 0.57}$	$96.60_{\pm 0.32}$	$78.62_{\pm 0.35}$	$7.18_{\pm 0.45}$	$3.60_{\pm 0.28}$	$24.44_{\pm 0.75}$	$9.80_{\pm 0.40}$
24	Spatter (S5) + Motion Bias (S)	✓	✓	$33.85_{\pm 0.82}$	$23.98_{\pm 0.61}$	$95.94_{\pm 0.34}$	$78.64_{\pm 0.36}$	$7.64_{\pm 0.46}$	$2.93_{\pm 0.23}$	$20.91_{\pm 0.71}$	$9.39_{\pm 0.40}$
25	Defocus Blur (S3) + Motion Drift	✓	✓	$89.72_{\pm 0.53}$	$65.84_{\pm 0.47}$	$94.84_{\pm 0.39}$	$75.97_{\pm 0.37}$	$14.16_{\pm 0.61}$	$6.26_{\pm 0.34}$	$23.56_{\pm 0.74}$	$10.65_{\pm 0.43}$
26	Defocus Blur (S5) + Motion Drift	✓	✓	$73.92_{\pm 0.76}$	$50.84_{\pm 0.59}$	$94.72_{\pm 0.39}$	$76.21_{\pm 0.37}$	$4.57_{\pm 0.36}$	$2.10_{\pm 0.21}$	$17.26_{\pm 0.66}$	$7.04_{\pm 0.35}$
27	Speckle Noise (S3) + Motion Drift	✓	✓	$86.65_{\pm 0.59}$	$62.44_{\pm 0.53}$	$93.99_{\pm 0.41}$	$75.02_{\pm 0.39}$	$13.46_{\pm 0.60}$	$5.95_{\pm 0.33}$	$23.01_{\pm 0.73}$	$9.96_{\pm 0.41}$
28	Speckle Noise (S5) + Motion Drift	✓	✓	$63.18_{\pm 0.84}$	$43.29_{\pm 0.65}$	$94.51_{\pm 0.40}$	$75.34_{\pm 0.38}$	$7.49_{\pm 0.80}$	$3.63_{\pm 0.46}$	$18.93_{\pm 0.68}$	$7.85_{\pm 0.36}$
29	Spatter (S3) + Motion Drift	✓	✓	$37.70_{\pm 0.84}$	$24.27_{\pm 0.57}$	$94.57_{\pm 0.39}$	$75.34_{\pm 0.38}$	$7.15_{\pm 0.45}$	$3.59_{\pm 0.27}$	$23.44_{\pm 0.74}$	$9.72_{\pm 0.40}$
30	Spatter (S5) + Motion Drift	✓	✓	$33.36_{\pm 0.82}$	$23.59_{\pm 0.60}$	$95.03_{\pm 0.38}$	$75.84_{\pm 0.37}$	$7.21_{\pm 0.55}$	$2.77_{\pm 0.28}$	$18.69_{\pm 0.68}$	$8.37_{\pm 0.38}$
31	Defocus Blur (S3) + PyRobot [14] (ILQR) Mul. = 1.0	✓	✓	$93.99_{\pm 0.41}$	$58.88_{\pm 0.40}$	$97.66_{\pm 0.26}$	$70.54_{\pm 0.32}$	$16.13_{\pm 0.64}$	$5.22_{\pm 0.28}$	$22.68_{\pm 0.73}$	$7.33_{\pm 0.32}$
32	Defocus Blur (S5) + PyRobot [14] (ILQR) Mul. = 1.0	✓	✓	$79.34_{\pm 0.71}$	$42.29_{\pm 0.49}$	$97.24_{\pm 0.29}$	$70.35_{\pm 0.33}$	$5.81_{\pm 0.41}$	$1.04_{\pm 0.11}$	$18.48_{\pm 0.68}$	$5.86_{\pm 0.29}$
33	Speckle Noise (S3) + PyRobot [14] (ILQR) Mul. = 1.0	✓	✓	$88.38_{\pm 0.56}$	$54.60_{\pm 0.49}$	$96.12_{\pm 0.34}$	$68.67_{\pm 0.35}$	$14.95_{\pm 0.62}$	$4.71_{\pm 0.26}$	$24.11_{\pm 0.75}$	$7.51_{\pm 0.32}$
34	Speckle Noise (S5) + PyRobot [14] (ILQR) Mul. = 1.0	✓	✓	$67.12_{\pm 0.82}$	$37.77_{\pm 0.57}$	$96.36_{\pm 0.33}$	$69.44_{\pm 0.34}$	$8.89_{\pm 0.50}$	$2.66_{\pm 0.20}$	$18.72_{\pm 0.68}$	$5.73_{\pm 0.29}$
35	Spatter (S3) + PyRobot [14] (ILQR) Mul. = 1.0	✓	✓	$40.70_{\pm 0.86}$	$18.26_{\pm 0.45}$	$96.09_{\pm 0.34}$	$68.25_{\pm 0.36}$	$8.31_{\pm 0.48}$	$1.76_{\pm 0.16}$	$23.17_{\pm 0.74}$	$7.76_{\pm 0.33}$
36	Spatter (S5) + PyRobot [14] (ILQR) Mul. = 1.0	✓	✓	$36.37_{\pm 0.84}$	$19.70_{\pm 0.51}$	$96.03_{\pm 0.34}$	$68.98_{\pm 0.36}$	$8.58_{\pm 0.49}$	$2.09_{\pm 0.17}$	$20.85_{\pm 0.71}$	$7.41_{\pm 0.33}$

Table 4. **PointNav and ObjectNav Performance.** Degradation in task performance of pretrained PointNav (trained for $\sim$ 75M frames) and ObjectNav (trained for $\sim$ 300M frames) agents when evaluated under `vis` and `dyn` corruptions present in RobustNav. PointNav agents have additional access to a GPS-Compass sensor. For visual corruptions with controllable severity levels, we report results with severity set to 5 and 3. Performance is measured across tasks of varying difficulties (easy, medium and hard). Reported results are mean and standard error across 3 evaluation runs with different seeds. Rows are sorted based on SPL values for RGB PointNav agents. Success and SPL values are reported as percentages. (V = Visual, D = Dynamics)

tion from severity 3 $\rightarrow$5). For ObjectNav, we find that for both `RGB` and `RGB-D` variants, performance decreases as with increasing severity of corruptions (3 $\rightarrow$ 5).

Performance Breakdown by Episode Difficulty. In Tables. 5 and 6 we break down performance of PointNav and ObjectNav agents by difficulty of evaluation episodes (based on shortest path lengths). We report results for a subset of `vis`, `dyn` and `vis+dyn` corruptions (mean across 3 evaluation runs under noisy actuations, wherever applicable). For PointNav `RGB` agents, we find that while performance is comparable across easy, medium and hard episodes under clean settings, under corruptions, navigation performance decreases significantly with increase in episode difficulty – indicating that under corruptions, PointNav-`RGB` agents are more successful at reaching goal locations closer to the spawn location. However, this is not the case for PointNav `RGB-D` agents, where the drop in performance with increasing episode difficulty is much less pronounced. For ObjectNav-`RGB` agents, we observe that performance (in terms of `SR` and `SPL`) drops as episodes become more difficult. For ObjectNav-`RGB-D` agents, although we find comparable `SPL` across episode difficulties in some cases, the trends are mostly the same – decreasing performance (in terms of `SR` and `SPL`) with increasing episode difficulty.

1.6. Visual and Dynamics Corruptions

RobustNav intends to be a *stepping stone towards* the larger goal of developing deployable robust agents. Our contribution in this work is to highlight the lack of robustness in existing navigation agents via an accessible benchmark that can continually evolve to integrate more sophisticated corruptions (so as to rigorously study this aspect). Our `vis` corruptions are based on common scenarios an agent might encounter in the real world (partly inspired

Increasing Episode Difficulty → / Corruption ↓	Easy		Medium		Hard	
	SR↑	SPL↑	SR↑	SPL↑	SR↑	SPL↑
POINTNAV-RGB						
1 Clean	99.64±0.18	82.80±0.38	99.36±0.24	84.21±0.47	97.91±0.43	83.34±0.54
2 Low Lighting	99.36±0.24	80.59±0.45	95.54±0.62	75.83±0.70	85.34±1.07	68.22±0.94
3 Camera Crack	94.10±0.71	75.81±0.70	80.05±1.21	62.15±1.06	70.49±1.38	52.44±1.14
4 Spatter	74.93±1.31	57.85±1.08	18.67±1.18	12.03±0.80	10.20±0.91	6.69±0.63
5 Speckle Noise + Motion Bias (S)	86.74±1.02	60.86±0.96	61.11±1.47	41.30±1.15	45.17±1.50	30.93±1.11
6 Spatter + Motion Bias (S)	72.48±1.35	53.25±1.08	18.85±1.18	11.99±0.79	10.11±0.91	6.63±0.62
7 Speckle Noise + Motion Drift	88.74±0.95	63.57±0.89	59.47±1.48	38.75±1.11	41.26±1.49	27.50±1.07
8 Spatter + Motion Drift	73.21±1.34	54.16±1.06	17.12±1.14	10.50±0.74	9.65±0.89	6.04±0.58
POINTNAV-RGBD						
9 Clean	99.55±0.20	82.36±0.41	99.45±0.22	85.38±0.47	98.72±0.34	87.27±0.40
10 Low Lighting	99.55±0.20	82.25±0.42	99.36±0.24	86.15±0.43	98.91±0.31	86.73±0.41
11 Camera Crack	99.27±0.26	81.79±0.43	97.18±0.50	83.19±0.59	91.53±0.84	79.45±0.80
12 Spatter	99.82±0.13	82.40±0.40	99.09±0.29	84.69±0.48	99.00±0.30	86.96±0.41
13 Speckle Noise + Motion Bias (S)	96.28±0.57	75.59±0.62	97.27±0.49	80.77±0.56	96.81±0.53	82.11±0.55
14 Spatter + Motion Bias (S)	96.46±0.56	76.02±0.62	94.99±0.66	78.61±0.67	96.36±0.57	81.29±0.59
15 Speckle Noise + Motion Drift	99.27±0.26	77.85±0.41	96.17±0.58	76.77±0.61	88.07±0.98	71.39±0.86
16 Spatter + Motion Drift	99.18±0.27	77.24±0.44	97.36±0.48	78.42±0.53	88.52±0.96	71.87±0.85

Table 5. **Breakdown of POINTNAV Performance Degradation by Episode Difficulty.** Degradation in task performance of pre-trained POINTNAV RGB and RGB-D agents (trained for ∼75M frames) for episodes of varying difficulties (based on shortest path lengths) when evaluated under `vis` and `dyn` corruptions present in ROBUSTNAV. For visual corruptions with controllable severity levels, severity is set to 5 (worst). Reported results are mean and standard error across 3 evaluation runs under noisy actuations (wherever applicable). Success and SPL values are reported as percentages.

from [10]) – ranging from additive noise, external/internal artifacts to effects of motion. These are implemented using open-sourced packages such as wand, open-cv & scipy. Speckle noise is implemented as per-pixel additive gaussian noise where the noise added is proportional to the pixel intensity in the ego-centric `RGB` frame. Motion-blur is implemented as a gaussian blur operation along a linear direction to simulate a motion effect. Spatter is implemented by overlaying occlusions (water droplets or particles on lens; distributed based on severity levels) on the ego-centric `RGB` frame. Defocus Blur is implemented via a gaussian blur filter to make objects in the ego-centric `RGB` frame out-of-focus. Low Lighting is implemented by first converting the `RGB` frame to the `HSV` color space and then modulating the intensities in the value channel. Camera-crack is the result of overlaying crack images over frame. Our `dyn` corruptions are motivated from and in line with well-known systematic and/or stochastic drifts (due to error accumulation) and biases in robot motion [13, 3, 7]. To periodically update the benchmark, we intend to conduct experiments akin to [11] to continually assess the Sim-2-Real predictivity of navigation performance under corruptions. For more details about the `vis` and `dyn` corruptions, please refer to our code at https://github.com/allenai/robustnav.

Increasing Episode Difficulty → / Corruption ↓	Easy		Medium		Hard	
	SR↑	SPL↑	SR↑	SPL↑	SR↑	SPL↑
OBJECTNAV-RGB						
1 Clean	40.50±1.94	12.43±1.04	33.48±1.29	15.51±0.73	25.75±1.21	14.49±0.75
2 Low Lighting	22.59±1.65	8.50±0.96	13.23±0.93	5.60±0.46	4.75±0.59	2.40±0.33
3 Camera Crack	21.65±1.63	10.10±1.05	5.38±0.62	2.72±0.34	1.61±0.35	1.15±0.26
4 Spatter	21.18±1.61	6.39±0.78	6.65±0.68	2.66±0.32	2.38±0.42	0.95±0.18
5 Speckle Noise + Motion Bias (S)	20.56±1.60	7.01±0.85	9.79±0.81	4.89±0.47	2.38±0.42	1.23±0.23
6 Spatter + Motion Bias (S)	20.56±1.60	6.71±0.83	7.32±0.71	3.35±0.38	1.61±0.35	0.65±0.15
7 Speckle Noise + Motion Drift	18.69±2.67	8.55±1.59	7.62±1.26	3.86±0.74	1.84±0.64	0.97±0.36
8 Spatter + Motion Drift	21.50±1.99	7.08±1.01	6.84±0.85	3.18±0.45	0.57±0.26	0.23±0.11
OBJECTNAV-RGBD						
9 Clean	46.73±1.97	15.22±1.15	35.72±1.31	18.09±0.80	29.58±1.26	18.18±0.86
10 Low Lighting	28.82±1.79	10.55±1.04	25.41±1.19	11.56±0.68	18.31±1.07	9.68±0.65
11 Camera Crack	35.51±1.89	11.53±1.03	28.03±1.23	13.90±0.73	22.45±1.16	14.03±0.79
12 Spatter	29.75±1.81	9.79±0.99	18.76±1.07	9.06±0.62	20.08±1.11	11.00±0.67
13 Speckle Noise + Motion Bias (S)	22.12±1.64	5.93±0.80	18.54±1.06	8.29±0.58	16.40±1.03	7.43±0.54
14 Spatter + Motion Bias (S)	27.26±1.76	8.66±0.92	19.81±1.09	8.81±0.60	18.93±1.08	10.35±0.66
15 Speckle Noise + Motion Drift	22.74±1.66	6.35±0.84	19.13±1.08	7.86±0.56	16.86±1.04	8.56±0.59
16 Spatter + Motion Drift	25.08±1.71	8.16±0.89	17.79±1.05	8.16±0.57	16.48±1.03	8.69±0.60

Table 6. **Breakdown of OBJECTNAV Performance Degradation by Episode Difficulty.** Degradation in task performance of pre-trained OBJECTNAV RGB and RGB-D agents (trained for ∼300M frames) for episodes of varying difficulties (based on shortest path lengths) when evaluated under `vis` and `dyn` corruptions present in ROBUSTNAV. For visual corruptions with controllable severity levels, severity is set to 5 (worst). Reported results are mean and standard error across 3 evaluation runs under noisy actuations (wherever applicable). Success and SPL values are reported as percentages.

References

[1] *Habitat Challenge*, 2020. https://aihabitat.org/challenge/2020/. 3

[2] 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*, 2020. 3

[3] Kostas E. Bekris, Andrew Ladd, and Lydia E. Kavraki. Efficient motion planners for systems with dynamics. In *ICRA Workshop*, 2007. 6

[4] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *CVPR*, 2015. 3, 4

[5] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, Luca Weihs, Mark Yatskar, and Ali Farhadi. RoboTHOR: An Open Simulation-to-Real Embodied AI Platform. In *CVPR*, 2020. 2

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2

[7] Carlos Garcia-Saura. Self-calibration of a differential wheeled robot using only a gyroscope and a distance sensor. *arXiv*, 2015. 6

[8] Nicklas Hansen, Yu Sun, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *ICLR*, 2021. 2

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[10] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019. 6

[11] 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. In *IROS*, 2020. 6

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[13] Andrew M. Ladd and Lydia E. Kavraki. Motion planning in the presence of drift, underactuation and discrete system changes. In *RSS*, 2005. 6

[14] 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*, 2019. 5

[15] Santhosh K. Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation, 2020. 3

[16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 2

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. 2

[18] Luca Weihs, Jordi Salvador, Klemen Kotar, Unnat Jain, Kuo-Hao Zeng, Roozbeh Mottaghi, and Aniruddha Kembhavi. Allenact: A framework for embodied ai research. *arXiv*, 2020. 2

[19] Erik Wijmans, Abhishek Kadian, Ari S. 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 *ICLR*, 2020. 2

[20] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, 2018. 3