International Journal of Semantic Computing

Language-Based Augmentation to Mitigate Shortcut Learning in Object-Goal Navigation

Dennis Hoftijzer
EEMCS, University of Twente, Drienerlolaan 5
Enschede, Overijssel, The Netherlands
dennishoftijzer@gmail.com

Gertjan Burghouts
Intelligent Imaging, TNO, Oude Waalsdorperweg 63
Den Haag, Zuid-Holland, The Netherlands
gertjan.burghouts@tno.nl

Luuk Spreeuwers
EEMCS, University of Twente, Drienerlolaan 5
Enschede, Overijssel, The Netherlands
l.j.spreeuwers@utwente.nl

Received (February 28, 2024)
Revised (Day Month Year)
Accepted (Day Month Year)

Deep Reinforcement Learning (DRL) has shown great potential in enabling robots to find certain objects (e.g., 'find a fridge') in environments like homes or schools. This task is known as *Object-Goal Navigation* (ObjectNav). DRL methods are predominantly trained and evaluated using environment simulators. Although DRL has shown impressive results, the simulators may be biased or limited. This creates a risk of *shortcut learning*, i.e., learning a policy tailored to specific visual details of training environments. We aim to deepen our understanding of shortcut learning in ObjectNav, its implications and propose a solution. We design an experiment for inserting a shortcut bias in the appearance of training environments. As a proof-of-concept, we associate room types to specific wall colors (e.g., bedrooms with green walls), and observe poor generalization of a state-of-the-art (SOTA) ObjectNav method to environments where this is not the case (e.g., bedrooms with blue walls). We find that shortcut learning is the root cause: the agent learns to navigate to target objects, by simply searching for the associated wall color of the target object's room. To solve this, we propose *Language-Based* (L-B) *augmentation*. Our key insight is that we can leverage the multimodal feature space of a Vision-Language Model (VLM) to augment visual representations directly at the feature-level, requiring no changes to the simulator, and only an addition of one layer to the model. Where the SOTA ObjectNav method's success rate drops 69%, our proposal has only a drop of 23%. Code is available at `https://github.com/Dennishoftijzer/L-B_Augmentation`

Keywords: Vision-based Navigation; Deep Reinforcement Learning; Vision-Language

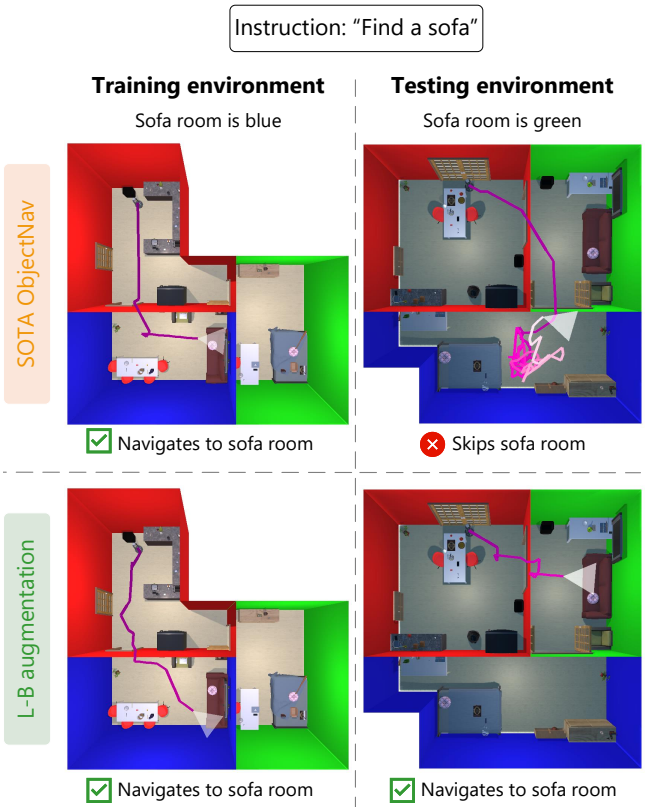


Fig. 1: We propose **Language-Based (L-B) augmentation** to generalize better to scenes with different wall colors. In this example, we interchange the wall color of the bedroom and living room, causing the SOTA objectNav method [1] to look for the sofa in the blue bedroom (wrong). With our augmentations, this is mitigated.

1. Introduction

Humans can easily find certain objects (e.g. 'find a fridge') in complex environments we have not seen before, such as a friend's house. We effortlessly avoid any obstacles but also reason about the unseen environment to decide which room to explore next e.g., 'where is a fridge most likely located?'. The embodied-AI (E-AI) community has made great strides by learning embodied agents (or 'virtual robot') such skills using Deep Reinforcement Learning (DRL) in a task called Object-Goal Navigation (ObjectNav) [2]. Despite good progress, DRL relies on gathering experience over millions (or billions) of iterations, making it impossible to learn in real-world environments. Therefore, research has been drawn to E-AI simulators [3–6], which allow for easily training agents in various simulated 3D indoor environments.

However, subtle, but detrimental, dataset biases might arise in E-AI simulators due to data collection artifacts, limitations in rendering, or simply unintended biases the simulator designer is not aware of. For instance, all kitchens in training environments might have a tiled floor. Consequently, training in E-AI simulators creates a profound risk of *shortcut learning* [7]: learning a simple, non-essential policy, tailored to specific details of the simulated environment, rather than learning any semantic reasoning or task-related skills. Efficient object-goal navigation involves learning useful semantic priors such as object-room relations (e.g., a fridge is in the kitchen), however can easily lead to unintended shortcuts (e.g., fridge is located near a tiled floor), which fail to generalize to environments where the shortcuts are no longer valid.

In this work, we deepen our understanding of shortcut learning in ObjectNav, its implications and propose a solution. First, we introduce an *out-of-distribution* (o.o.d.) generalization test. We insert a dataset bias in the appearance of training environments, which offers the agent a shortcut pathway for finding a given target object. As a proof-of-concept of such a shortcut bias, we associate each room type to a unique wall color i.e., kitchens have red walls, bedrooms have green walls and so forth. Using our setup, we are able to evaluate o.o.d. generalization of a state-of-the-art (SOTA) ObjectNav method [1] to environments where we change wall colors (e.g. kitchens now have blue walls). As a result, we find that (1) only changing wall colors degrades performance significantly, and (2) shortcut learning is the root cause. The agent learns to navigate towards target objects by simply searching for the wall color associated with the target object's room.

Secondly, we wish to bring more insight as to why the agent learns such a shortcut strategy by further analyzing the visual representations within the agent's architecture. We train a simple neural classifier on visual representations extracted from sampled RGB observations to classify room type. We train on biased combinations of room type-wall color and observe the classifier's predictions on held-out combinations. We find that the predictions are heavily biased towards wall color. Evidently, wall color serves as an unintended shortcut predictor for room type.

Finally, we aim to increase domain generalization. Domain randomization methods e.g., randomizing textures, colors and shapes of objects or environments are commonly used to transfer policies in DRL [8–10]. However, these methods specifically require changes to the simulator, which might be inflexible or difficult to modify e.g., high-fidelity simulators with training data reconstructed from real-world 3D scans [11, 12]. While more sophisticated methods for partially editing individual frames during training exist (e.g., text-to-image models [13, 14]), they are slow, computationally expensive and error-prone. Instead, we take a different approach and propose *Language-Based (L-B) augmentation* (see Fig. 1). We augment directly at feature-level, without editing individual frames or any changes to the simulator.

We build upon promising results from [1], where visual representations within the agent's architecture are based on a Vision-Language Model (VLM). RGB observations are encoded using a Contrastive Language Image Pretraining (CLIP) [15]

visual backbone. CLIP jointly trains an image and text encoder, such that both produce similar representations for visual concepts in images or their names in natural language. Our key insight is that we can augment agent's visual representations at feature-level, by describing variations of the dataset bias in natural language. By an elegant modification to the SOTA architecture [1], with only one additional layer, we generalize better to environments with different wall colors in ObjectNav.

2. Related work

2.1. *Vision-Language for visual navigation*

Several recent works have proposed utilizing pre-trained features of Vision-Language Models (VLMs), pre-trained on internet-scale data, for several visual navigation tasks [1, 16–19]. In [1], authors explore the effectiveness of learning a navigation policy based on CLIP embeddings [15]. With their method, EmbCLIP [1], they show that CLIP's visual representations encode useful navigation primitives such as reachability and object localization. They set new SOTA results on several visual navigation tasks, including ObjectNav, and show promising results for generalizing to an open-world setting i.e., navigating to target objects not seen during training. Moreover, as generating the robot trajectories and paired language annotations in the real world might be costly, further works have proposed utilizing VLMs out-of-the-box with maps to enable zero-shot navigation i.e., without supervision of DRL reward signals or human demonstrations [16–18]. We adopt the architecture of EmbCLIP as a baseline, given its strong performance on a variety of settings. However, our focus is specifically on addressing shortcut bias in E-AI simulators, which might impede generalization of DRL methods to novel environments.

2.2. *Embodied AI simulators and scene datasets*

Many Embodied-AI simulators [3–6] have been developed, along with several (near) photo-realistic 3D indoor scene datasets [10–12, 20]. Scenes can be either reconstructed from 3D scans of real-world houses e.g., Matterport [11], or synthetically composed from artist created 3D assets e.g., AI2-THOR [3] (The House Of inteRactions) and variants (RoboTHOR [5], ManipulaTHOR [21]). Both methods are extremely costly to collect. Reconstructing scenes from 3D scans involves stitching images from specialized cameras whilst manually composing synthetic scenes involves carefully configuring lighting, object placement and textures. ProcTHOR [10] recognizes this fact and instead uses a procedural generation process to generate 10,000 scenes (dubbed ProcTHOR-10k). In this work, we leverage the ProcTHOR-10k scene dataset. Due to the procedural generation process, we can fully customize these scenes by altering the appearance of individual objects and room surfaces (walls, floors and ceilings). For instance, a red sofa can be replaced with a black one. This customization and our proposed interventions on ProcTHOR-10k, allow for inserting a shortcut bias in the appearance of training scenes and evaluate o.o.d. generalization.

Although ProcTHOR enables E-AI to scale, this does not imply shortcut biases completely disappear. Recent works show that even Large-Language Models (LLMs), pre-trained on text amounting to billions of words, suffer from shortcut learning, largely due to collection artifacts in training data [7, 22]. Moreover, shortcut bias might be difficult to observe for humans e.g., superficial statistics in training data such as textures of specific frequencies in image classification tasks [23]. Consequently, in E-AI simulators, shortcut bias might arise inadvertently. For instance, as ProcTHOR is generated procedurally, some smaller objects (e.g. a pen) are always placed on larger objects (e.g. a desk in the bedroom). An agent might learn a bias for navigating to a target object only when it is placed on this larger object and not when the object is placed independently (e.g. on the floor in the living room). We employ a simple wall color bias as a proof-of-concept for such unintended shortcut biases.

2.3. *Shortcut learning*

Shortcut learning is emerging as a key impediment in the generalization ability of deep neural networks (DNNs) [7]. Shortcuts are decision rules, often learned by DNNs, which aid performance on a particular dataset but do not match with human-intended ones. Accordingly, they typically fail when tested in only slightly different conditions. Prior work in shortcut learning is predominantly concerned with supervised learning [23–25]. Similar to our work, [24] designs an experimental setup to observe whether DNNs prefer to adopt color, shape or size shortcuts, and find DNNs naturally prefer certain shortcuts. In contrast, we study the shortcut learning phenomenon in the context of DRL.

A common implication of shortcut learning in DRL is observed when transferring policies from simulation to the real-world [7, 8, 26]. Most policies trained in simulation generalize poorly to the real-world due to agents adapting to specific visual details of the simulator. Prior works cope with this so-called 'reality gap' by domain randomization methods i.e., randomizing appearances in training environments [8, 9]. Similarly, ProcTHOR [10] allows for randomizing e.g., textures and colors of walls, ceilings, floors and objects. While ProcTHOR shows incredibly powerful results, such augmentations might not be available for all simulators, and more often than not, difficult to apply post-hoc. Contrary, our method can readily be applied post-hoc as it requires no changes to training data or the simulator. We propose augmentations where we use targeted randomization of specific unintended biases, in our case, wall color. Although a simple wall color bias might be addressed using conventional domain randomization, these methods are inconvenient considering more intricate biases (e.g., a pen is always on a desk). In contrast, our method utilizes free-form natural language, which allows for easily adapting to different biases. Vision-Language Models (VLMs) e.g., CLIP [15], allows us to augment at feature-level based on prior knowledge of the environment without any changes to training data.

3. ObjectNav preliminaries

3.1. *ObjectNav definition*

In ObjectNav [2], agents are initialized at a random pose in an unseen environment and given a label specifying the target object category (e.g. 'Bed'). The goal for the agent is to navigate to an instance of the target object within a certain time budget ($T = 500$). At each time step t, the agent receives an image from an RGB forward-facing camera and can take one of 6 possible actions: move ahead, rotate left, rotate right, look up, look down and done. We do not utilize any depth sensor readings. Also, we simulate actuation noise to better resemble actuation in the real-world. A full description of the discrete action space is shown in Table 4 (Appendix).

3.2. *Evaluation metrics*

Following standard ObjectNav procedure [2], an episode is considered successful if (1) the agent executes the special done action; (2) The target object is within a certain distance threshold, typically $d_t = 1$m; and (3) the target object is considered visible i.e., within the camera's field of view and not fully obstructed.

We report two primary performance metrics: Success and Success weighted by (normalized inverse) Path Length (SPL). Success is the average success rate over all N evaluation episodes and SPL is a measure for path efficiency [2]:

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

S_i is a binary indicator denoting success of episode i; l_i is the shortest path length from starting position to the target object and p_i is the length of the path the agent travelled. SPL is bounded by $[0, 1]$, where 1 is optimal performance i.e., the agent took the shortest path possible in all N evaluation episodes. Note that, SPL is a stringent measure. Achieving an SPL of 1 is infeasible (even for humans), without knowing the target object location a priori. Additionally, we report two more evaluation metrics: Distance To Target (DTT) and the episode length. DTT is the remaining shortest path length to visibly see the target object.

3.3. *ProcTHOR*

ProcTHOR enables E-AI to scale by procedurally generating simulated environments. Given a room specification (e.g. a house with 1 bedroom and 1 bathroom), ProcTHOR can produce a large variety of floor plans, populates each floor plan by sampling from a library of 3D assets, and supports randomization of lighting, colors and textures. In this work, we leverage ProcTHOR-10k, as these scenes can be fully customized. This customization and our proposed interventions, allow for inserting a shortcut bias in the appearance of training environments and evaluate o.o.d. generalization.

4. O.o.d. test: interventions on ProcTHOR-10k

4.1. *Interventions on ProcTHOR-10k*

In order to evaluate o.o.d. generalization, we need to guarantee only to measure performance degradation due to changing wall colors, without other aspects (object appearances, scene layout, etc.) influencing our evaluation. Therefore, we propose some interventions on ProcTHOR-10k. First, we start by selecting a more uniform subset of scenes and targets. We select only houses with 3 rooms, which all consist of 3 room types: kitchen, bedroom and living room. For each room type, we select 3 target object categories (9 total) which are semantically related (e.g. fridge in kitchen). Section Appendix A (Appendix) shows an overview of all target objects selected. Next, we restrict ourselves to scenes which contain exactly one instance of each target object category in the associated target room e.g., every house contains 1 bed, positioned in the bedroom. We ensure this restriction by (1) selecting scenes which contain at least one instance of each target object in the associated room type and (2) manually removing any double (or more) instances of target objects. Secondly, we set identical appearances for all object categories (including doors) e.g., all chairs appear exactly alike, and identical appearances of room surfaces for each room type e.g., all kitchens have identically colored walls, floors and ceilings. We set object appearances identical by assigning one 3D asset from the ProcTHOR library to each object type. We set all room surfaces identical by setting the same materials from the ProcTHOR library. Lastly, we remove windows and wall decoration. These interventions limit the house variations to just wall colors, which is the only aspect influencing the performance.

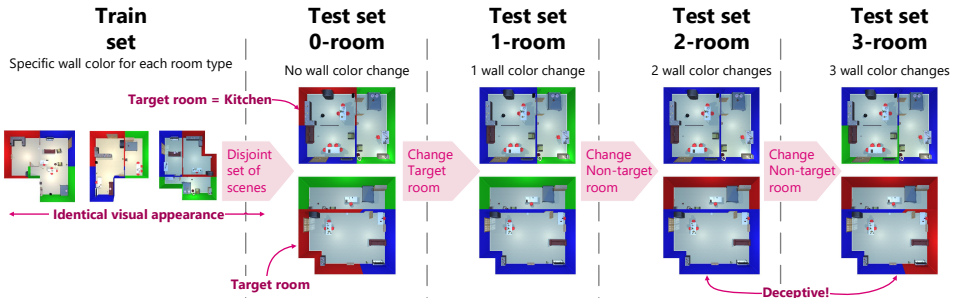


Fig. 2: **Setup for our o.o.d. generalization test.** In this example, the target room is the kitchen (red walls in test set 0-room). We change the target room first (test set 1-room) and incrementally change more rooms (test set 2/3-room). The bottom row shows two examples of deceptive changes, where the wall color associated with the target room (red wall color) is moved to a different room type. The top row only shows nondeceptive changes.

4.2. *Evaluating o.o.d. generalization*

We aim to assess the agent's generalization across increasing numbers of changing wall colors (illustrated in Fig. 2). The training set consists of visually identical houses, where living rooms have blue walls, kitchens have red walls and bedrooms have green walls. For our test sets, we use houses with a different layout such that agents cannot simply memorize object locations, and permute wall colors. The 0-room test set serves as a reference. To solely evaluate generalization to different wall colors, we use the same layouts in each test set and compare performance to the reference 0-room test set. First, we change the wall color of the target object's room as this is the simplest deviation (1 wall color change). For instance, if the target object is a fridge, we start by altering the wall color of the kitchen from red to e.g., green, whereas if the target object is a bed, we start with changing the wall color of the bedroom from green to e.g., blue. Next, we change another room's wall color (2 wall color changes). Finally, we change the wall colors of all three rooms (3 wall color changes). We change to all possible permutations (e.g. red kitchen to blue and green wall colors in test set 1-room) with repetition i.e., multiple room types can have the same wall color.

We differentiate wall color changes of non-target rooms (test set 2- and 3-room) into two types: *deceptive* vs *nondeceptive*. We expect that moving the learned color i.e, the wall color associated with the target room, to a non-target room will have a high impact, because the agent may look in the latter, wrong room. We refer to this wall color change as 'deceptive'. Examples of deceptive changes are shown in the bottom row (test set 2- and 3-room). Instead, when none of the rooms has the learned color, we expect less performance degradation, because the agent is not misled. We refer to such a wall color change as 'nondeceptive' (top row).

5. Method: Language-based Augmentation

We increase domain generalization by augmenting the agent's visual representations at feature-level, such that these are more invariant to changing environments. We implement this by adding one layer on top of EmbCLIP [1]. In EmbCLIP, the agent's visual representations are based on a VLM (CLIP). We leverage the vision-language representations for feature-level augmentations, without the need to modify the simulator. Our augmentations are based on textual descriptions of variations of the dataset bias that we want the agent to learn and generalize. We call this *Language-Based (L-B) augmentation* (Fig. 3). In EmbCLIP, at each time step t, a visual representation or image embedding $\boldsymbol{I_t}$ is obtained by encoding RGB observations using CLIP's [15] visual encoder (CLIP_v). CLIP learns to associate text strings with their visual concepts in images. Our key insight is that we can represent domain specific knowledge, regarding the changes in environment appearances, using natural language. By encoding text descriptions of variations of the dataset bias (e.g. 'a blue wall'), using CLIP's text encoder (CLIP_T), we vary visual representations without actually having seen images containing these variations (e.g. an image of a blue

wall). This allows us to augment directly at feature-level. For encoding the text descriptions we use the default prompt template recommended by [15]: 'a photo of a {label}'. We insert descriptions of variations of the dataset bias (e.g. 'red wall'). We obtain our augmented embeddings $\boldsymbol{I_t^{LB}}$ by computing differences between n encoded text descriptions of variations of the dataset bias $\boldsymbol{T_{1,\ldots,n}}$, and adding to visual representation $\boldsymbol{I_t}$:

$$\boldsymbol{I_t^{LB}} = \boldsymbol{I_t} + \alpha \cdot \Delta(\boldsymbol{T}), \tag{2}$$

$$\Delta(\boldsymbol{T}) = \begin{bmatrix} \boldsymbol{\Delta_1} \\ \boldsymbol{\Delta_2} \\ \vdots \\ \boldsymbol{\Delta_{n(n-1)}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{T_1 - T_2} \\ \boldsymbol{T_1 - T_3} \\ \vdots \\ \boldsymbol{T_n - T_1} \end{bmatrix} \tag{3}$$

where, α controls the degree of augmentation and Δ computes differences of all permutations of length 2 of text descriptions $\boldsymbol{T}$. By randomly sampling an augmented embedding from $\boldsymbol{I_t^{LB}}$ at each time step, we aim to provide the RL model (RNN) with an embedding which resembles the same room type (e.g., a living room in Fig. 3), but with a different wall color (e.g., red and blue instead of green in Fig. 3). We empirically find $\alpha = 50$ to work well by tuning for our specific dataset and shortcut bias. We standardize features before feeding into the RNN to ensure

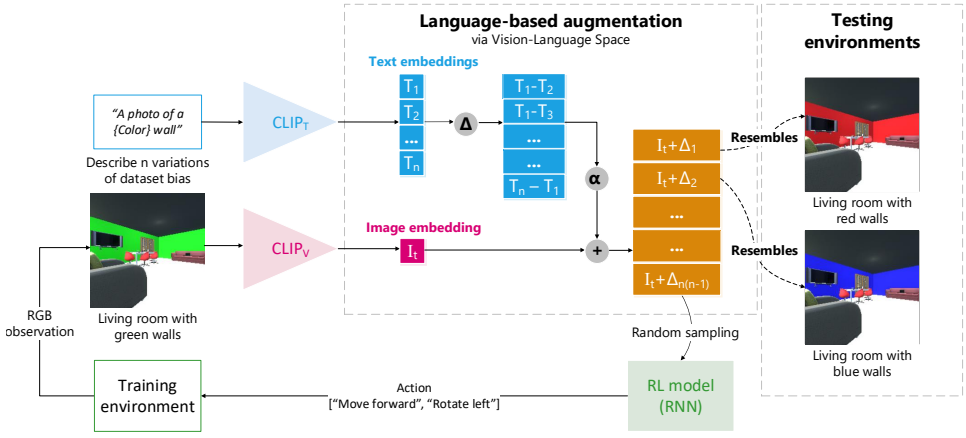


Fig. 3: **Language-Based (L-B) augmentation via a the feature space of a vision-language space.** Our key insight is that we can augment agent's visual representations (I_t) using differences (Δ) between encoded text descriptions of variations of the dataset bias ($T_{1,\ldots,n}$). The augmented embedding of an image 'A living room with green walls' resembles a 'living room with red or blue walls'. The RL model (RNN) is not able to use a shortcut strategy even if during training living rooms always have green walls.

stability during training (as some features might dominate the loss due to large norms). In our case, we insert three ($n = 3$) text descriptions of variations of the dataset bias: 'blue wall', 'red wall' and 'green wall'. This results in 6 augmented embeddings $\boldsymbol{I_t} + \boldsymbol{\Delta_{n(n-1)}}$ per image embedding $\boldsymbol{I_t}$.

6. Experiments

We perform four experiments, where we aim to: (1) evaluate generalization of EmbCLIP [1] to scenes where we change wall colors and study to what extend shortcuts influence the generalization ability; (2) bring more insight into why the agent learns shortcuts by analysing visual representations within EmbCLIP; (3) demonstrate, using an identical o.o.d. analysis, that our L-B augmented representations mitigate the effects of shortcut bias and finally; (4) validate L-B augmentation can increase domain generalization in ObjectNav by integrating within the architecture of EmbCLIP.

6.1. *Experimental setup*

6.1.1. *ObjectNav dataset details*

We train agents on 20 visually identical but biased scenes, generated using our proposed interventions (Section 4). During training, we randomly sample 1 of 9 target objects. We analyze o.o.d. performance on a set of 5 disjoint scene layouts. We run 1080 evaluation episodes per test set to evenly distribute episodes over the 5 layouts, possible wall color permutations and target objects. Section Appendix B (Appendix) details the distribution.

6.1.2. *Agent architecture and configuration*

We use the ObjectNav EmbCLIP architecture [1], which has two different variations: a closed-world architecture, which assumes known target objects, and an open-world variant. Both encode RGB egocentric views using a frozen CLIP image encoder with a ResNet-50 backbone. However, the closed-world variant obtains a goal-conditioned embedding before feeding into an RNN, which involves removing the final layers from CLIP, whereas the open-world variant feeds the image embedding directly. For our o.o.d. generalization test, we adopt EmbCLIP's closed-world architecture given its better performance in a closed-world setting. We integrate our L-B augmentations in the open-world variant, as this architecture feeds the CLIP image embedding directly into an RNN, which allows for substituting this image embedding with a L-B augmented embedding, using random sampling.

Following typical ObjectNav setup [1, 10], the embodied agent approximately matches a LoCoBot with a 90° horizontal camera field of view, a $0.25m$ step size and a 30° turn angle.

6.1.3. *Reward setting*

At each time step t, the reward r_t is:

$$r_t = max(0, min\Delta_{0:t-1} - \Delta_t) + r_{slack} + r_{succ} \tag{4}$$

where, $min\Delta_{0:t-1}$ is the minimal path length from the agent to the target object that the agent has previously observed during the episode in $\{0, ..., t-1\}$, Δ_t is the current path length from the agent to the target, $r_{slack} = -0.01$ is the slack penalty, and $r_{succ} = 10$ is a large reward if the episode is considered successful, otherwise $r_{succ} = 0$. The reward is shaped to optimize path efficiency and, therefore, SPL.

6.1.4. *Implementation details*

We use the Allenact framework [27] and render frames at 224×224 resolution. To parallelize training, we use DD-PPO [28] with 40 environment instances. After each rollout, the model is updated using 4 epochs of PPO [29] in a single global batch size of 7680 frames. We perform validation every 200,000 frames and report results of the checkpoint with highest SPL. Additional hyperparemeters are shown in Section Appendix C (Appendix).

6.2. *Impact of Shortcut Learning*

How well does a SOTA ObjectNav method generalize to scenes with different wall colors and to what extent do shortcuts affect the o.o.d. generalization ability? We hypothesise the performance drop coheres with the number of wall color changes. Moreover, we posit a deceptive change will lead to more performance degradation than a nondeceptive change as the agent will be misled to search for the target object in the wrong room. Fig. 4 shows the performance. We report the mean over all episodes, the mean over episodes with deceptive changes and the mean over episodes with nondeceptive changes. We observe that changing the wall colors of only the target room already leads to a large decrease in performance. On average (blue mean bar), we observe a 67% relative drop in SPL ($0.39 \rightarrow 0.13$) and 56% in success rate ($68\% \rightarrow 30\%$) going from 0 wall color changes to 1 wall color change, with even lower mean performance for more wall color changes. Indeed, we find that EmbCLIP generalizes poorly to scenes with different wall colors when using limited training data. Regarding deceptive vs nondeceptive changes, the Success, SPL and DTT metrics indicate that deceptive changes indeed deteriorate performance most, even more than multiple nondeceptive changes. Interestingly, however, we observe shorter episode lengths for deceptive changes. We conjecture that due to deceptive changes, the agent directly navigates towards the learned color of the target room, which is now placed in a non-target room, without exploring any other rooms. The agent will erroneously search this non-target room, but can not find the target object, and terminates the episode. In contrast, an agent will explore the entire scene when wall colors have only been changed nondeceptively, leading to longer

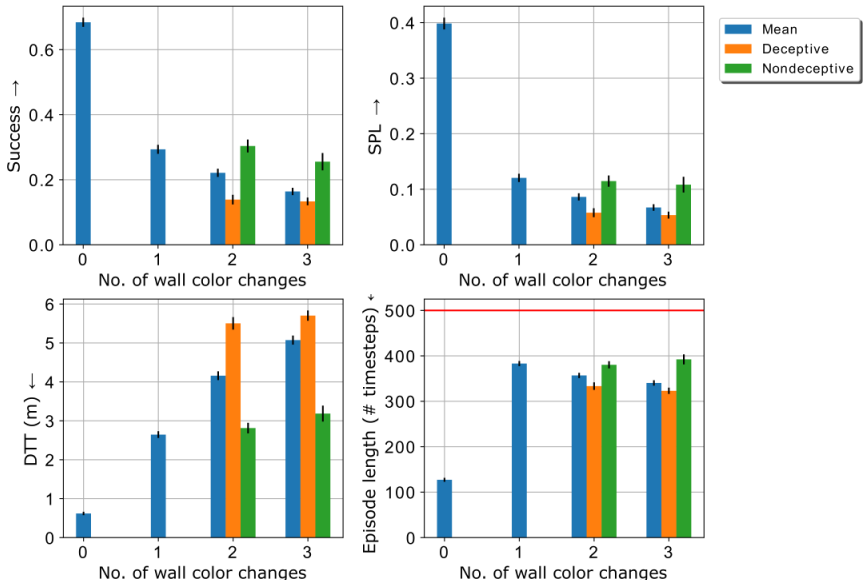


Fig. 4: **Degradation for o.o.d. cases.** Performance of EmbCLIP [1] to scenes with different wall colors. When only changing the wall color of the target object's room (1 wall color change), we already observe a large decrease in performance in all metrics.

episodes. We show a qualitative example of this behaviour in Fig. 5, where the agent has learned to look for a room with green walls instead of a bedroom. The agent is deceived by the green living room, and terminates the episode when it sees the sofa. This leads to a much shorter episode than in the nondeceptive example, where the agent explores large parts of the scene. Evidently, the agent has learned a shortcut strategy, it navigates towards a particular wall color instead of the right target room.

6.3. *Analysis of agent's visual representations*

Why does the agent learn such a shortcut strategy in ObjectNav? We aim to understand if we can trace this issue back to the agent's visual encoder (CLIP_v). Specifically, we posit wall color serves as an unintended shortcut predictor for room type in the agent's visual representations. To this end, we train a simple neural classifier, using supervised learning, to predict room type based on CLIP embeddings extracted from sampled RGB observations. We evaluate if the classifier adopts wall color as a shortcut predictor by observing its predictions on held-out room type-wall color combinations.

To enable the above evaluation, we render a small dataset of sampled frames. Specifically, we select 1 house layout, randomly position an agent in a certain room,

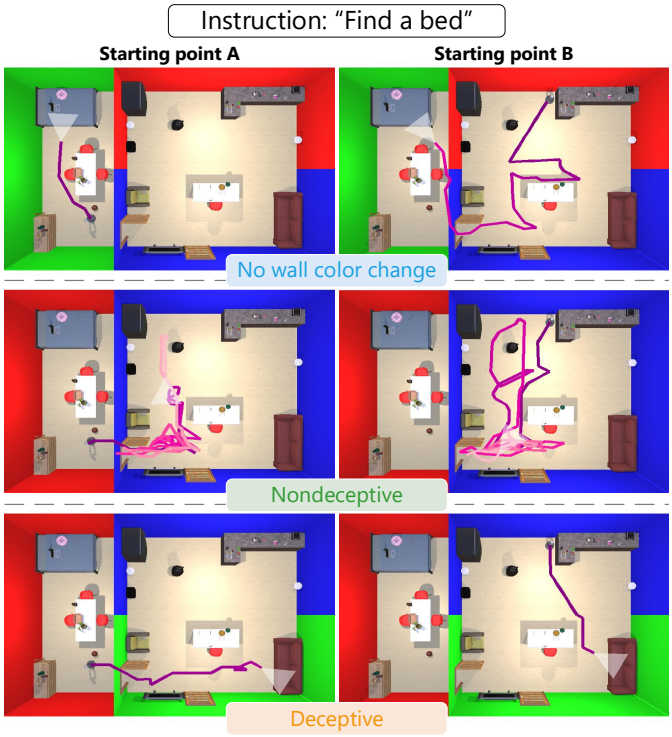
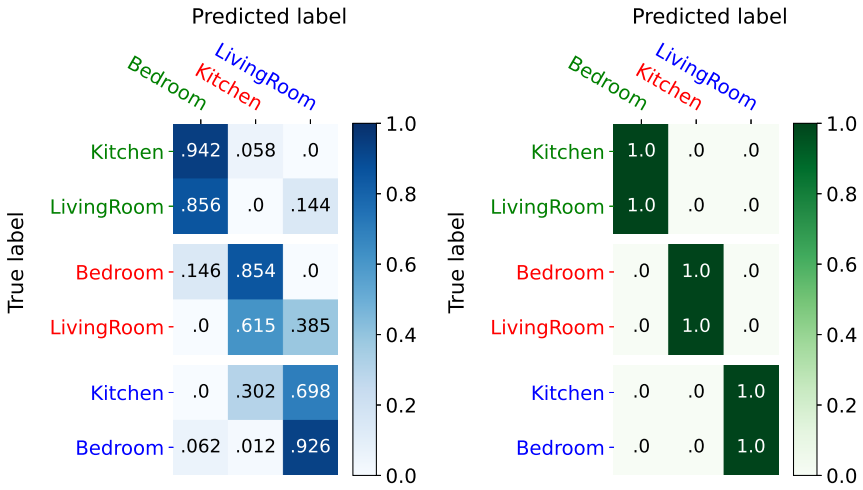


Fig. 5: **Errors and shortcuts by the SOTA ObjectNav method.** We show example trajectories from 2 different starting position (left vs right column). Notice how nondeceptive episodes (middle) are much longer than deceptive episodes (bottom), whilst both are unsuccessful. Also note the absolute lack of search in the bedroom when changing wall colors deceptively.

and sample its RGB obervation. We sample 200 observations from each room type to form our training set. Ground truth data is determined by the agent's location. Next, to form our test set, we permute wall colors and sample 200 more frames for every held-out combination of room type–wall color. For each frame in our dataset, we extract CLIP embeddings using CLIP's visual encoder. We train a multi-layer perceptron (MLP) to predict room type solely from these embeddings. Furthermore, we split our test set into two types: 'context' vs 'contextless'. We refer to 'context' frames as frames which contain objects, which provide semantic information (or 'context') pertaining to room type (e.g. a sofa in the living room). Instead, 'contextless' frames do not contain any useful information (i.e. only walls, floors and ceilings). We create this split as we expect the predictions to be worse on contextless frames as the classifier can only use the biased wall color to classify room type. See Appendix (Section Appendix D) for more details on the frame sampling and classifier training.



(a) Context (b) Contextless

Fig. 6: **Room type confusion CLIP visual representations.** Confusion matrices for a classifier trained on CLIP embeddings extracted from wall color biased frames. True label colors indicate the wall color in our test set. Predicted label colors indicate the room type–wall color combinations during training.

Fig. 6 shows confusion matrices for (a) context and (b) contextless frames. Fig. 6a shows that e.g., 94.2% of frames showing a kitchen with green walls, are classified as a bedroom since the classifier has only seen bedrooms with green walls during training. The majority of context frames are being classified directly according to their wall color, leading to classification accuracy of 16.8%, worse than random. In the contextless set (Fig. 6b), we see all rooms being classified according to their wall color leading to an overall classification accuracy of 0%. Clearly, we observe a large confusion of room types, which is caused by the biased wall color.

6.4. *Analysis of L-B augmented representations*

In the previous experiment we observed that CLIP visual representations encode shortcuts where wall color serves as an unintended shortcut predictor for room type. We posit that improved room type classification leads to more capable ObjectNav agents in our o.o.d. generalization test. Therefore, in the following experiment, we investigate the impact of augmenting CLIP visual representations, using our L-B augmentation (Section 5), on room type classification accuracy.

Although an agent cannot base its room type predictions on wall color, it can base its predictions on object-room relations. Therefore, using L-B augmentation, we aim to improve classification accuracy on context frames, whereas, when no context is provided, the agent should not recognize the room type. Hence, we aim to

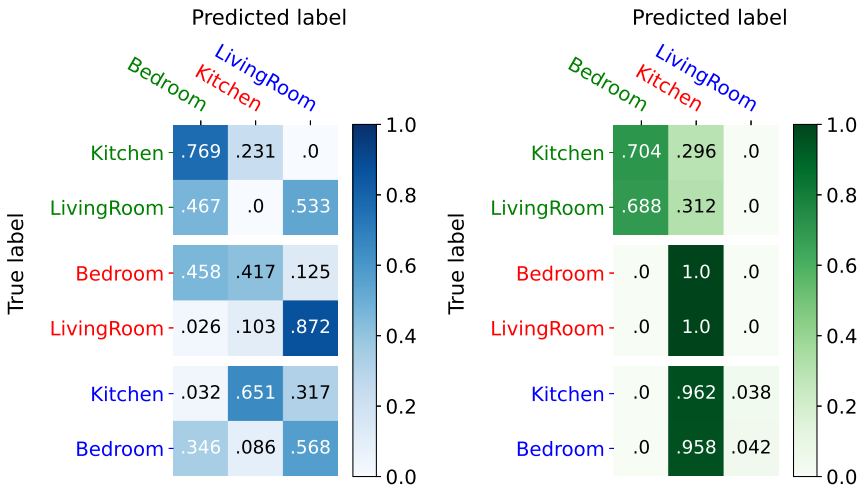
achieve unbiased confusion on contextless frames i.e., a confusion matrix $\boldsymbol{C}$ where each element $C_{i,j} = 0.33$ (for $n_{classes} = 3$). To measure unbiased confusion, we define the Absolute Difference With Random (ADWR):

$$\text{ADWR} = \frac{1}{N} \sum_{i,j} |C_{i,j} - \frac{1}{n_{classes}}| \tag{5}$$

where, $N = 18$ is the number of elements in the confusion matrix $\boldsymbol{C}$. Ideally, ADWR $= 0$, where the classifier predicts each room type with equal probability. Contrary, when predictions are completely biased i.e., predicting all test samples according to the learned wall color, ADWR $= 0.44$. Note that ADWR $\in [0, 0.44]$.

We use an identical experiment setup as in the previous experiment (Section 6.3). However, now we first augment the CLIP embeddings extracted from the frames in our training set according to Eq. 2 and 3. As this results in six times as many L-B representations, we randomly sample a single L-B representation per frame for our new train set. We test on the original CLIP embeddings in our test set, where again we split in context and contextless frames, and report confusion matrices.

Fig. 7a shows improved room type classification on context frames using our L-B augmentations. Although the bias is still significantly influencing the room type prediction, we see, for instance, 23.1% of green kitchen frames now being classified as their true type, opposed to only 5.8% in Fig. 6a. Surprisingly, in Fig. 7b, we see



(a) Context (b) Contextless

Fig. 7: **Room type confusion after L-B augmentations.** Classifier trained on L-B augmented representations. Label color indicates wall color, where the predicted label colors indicate the wall colors the classifier was trained on.

Table 1: **Original CLIP visual representation vs after Language-Based augmentations.** We report accuracy on the context frames and ADWR (see text) on the contextless frames.

	Accuracy (↑) (Context)	ADWR (↓) (Contextless)
Original CLIP	0.17	0.44
Language-based augmentation (ours)	**0.50**	**0.37**

a shift towards predicting contextless frames as a kitchen type instead of unbiased confusion. We posit this is due to optimizing classification accuracy during training using log-loss, which indicates how far predictions are from ground truth. There is no incentive to classify rooms randomly, with a low probability, over classifying all test samples as a single class. Both result in low classification accuracy. Hence, there seems to be a mismatch between the optimization and what we aim to achieve.

In Table 1, we compare results between a classifier trained on the original CLIP embeddings vs after L-B augmentions. Using our method, we observe significantly increased classification accuracy on context frames (17% → 50%), as desired. Furthermore, we see decreased ADWR on contextless frames (0.44 → 0.37), as desired. Overall, using L-B augmentation we are able to suppress the use of wall color serving as a shortcut predictor for room type classification.

6.5. *Benefit of Proposed L-B Augmentations for navigation*

Do our L-B augmentations make the agent's model more robust to shortcuts, i.e., more domain invariant against biased wall color and room type? Finally, we integrate our L-B augmentation method within the EmbCLIP architecture as a single extra layer, as detailed in Section 5. Note that training time is only marginally longer: from 88 to 90 GPU-hours (both 30M steps). Fig. 8 shows a comparison of EmbCLIP with and without our L-B augmentations. EmbCLIP's performance already degrades significantly after changing wall colors of the target room (1 wall color change). We observe 69% relative drop in success (45% → 14%) and 82% drop in SPL (0.22 → 0.04). In contrast, our method shows improved domain generalization. When changing wall colors of the target room (1 wall color change), our method incurs only a 23% relative drop in success (39% → 30%) and 29% drop in SPL (0.17 → 0.12). We observe less performance degradation with an increasing number of wall color changes than EmbCLIP. In the qualitative example of Fig. 1, the agent is now able to find the sofa even though it is not in a living room with blue walls (training). The agent finds the sofa successfully in a living room with green walls (not seen during training). These results demonstrate that our L-B augmentations are an interesting direction to make RL agents more robust to biases, by only adding one additional layer to the agent's model.

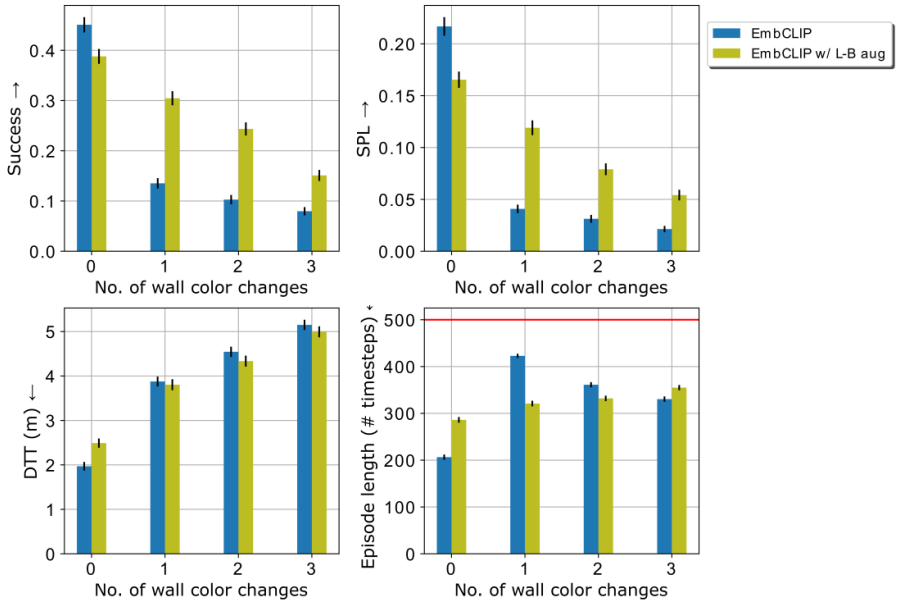


Fig. 8: **Extending EmbCLIP [1] with our L-B augmentations.** Performance on o.o.d. cases for EmbCLIP [1] vs EmbCLIP with L-B augmentations. Using our L-B augmentations, EmbCLIP generalizes better to scenes with different wall colors.

7. Conclusion and limitations

We evaluated how well a SOTA method for ObjectNav generalizes to scenes with different wall colors, and studied to what extent shortcut learning influences this o.o.d. generalization. We found that, when deliberately limiting training data, only changing wall colors in testing scenes decreases performance significantly, with the root cause being the deceptive wall color changes. We proposed Language-Based (L-B) augmentation to mitigate shortcut learning. By encoding text descriptions of variations of the dataset bias, and leveraging the multimodal embedding space of CLIP, we were able to augment agent's visual representations directly at feature-level. Finally, experiments showed that our L-B augmentation method is able to improve domain generalization to scenes with different wall colors in ObjectNav. When changing the target object's room, our method incurs a 23% relative drop in success rate whilst the SOTA ObjectNav method's success rate drops 69%.

To demonstrate the usefulness of our approach, we considered a simple case of shortcut bias e.g., wall color. Although our agents showed improved domain generalization, such simple cases may well be accommodated using conventional domain randomization methods in simulators which are easily modifiable. In future work, we hope to explore how to use natural language for augmentations addressing

more intricate biases. For instance, bias at the object-level. Some objects might usually occur in combination with other objects (e.g., pillow on a bed). Agents could learn a bias for navigating towards the bedroom instead of also exploring the living room (e.g., pillow on a sofa). Moreover, as efficient ObjectNav requires agents to leverage useful semantic priors about the environment (e.g., object-room relations), it would be interesting to see how to use natural language to guide the exploration of agents in more unusual situations, where such priors are disadvantageous (e.g. pillow in the kitchen).

Appendix A. Target object selection

We select 3 target objects per room type, based on the object's occurrence frequency in ProcTHOR-10k and if they have a clear semantic relation with one of the room types. We select different sized objects for each room type (see Table 2).

Appendix B. Distribution evaluation episodes

We evenly distribute 1080 evaluation episodes over 9 target objects, 5 scene layouts and possible wall color permutations for each test set. As there are more possible permutations for increasing number of wall color changes, the number of episodes per unique scene (combination of layout and wall color permutation) decreases. For example, for the 0-room test set, only 5 unique scenes are possible as only 1 wall color permutation is possible (no wall color changes w.r.t. training set). Hence, we run 216 episodes per unique scene. For the 1-room test set, we change the target room to 2 different wall colors e.g., bedroom from green (train) to blue or red. In this case, we distribute the 1080 episodes over 10 unique scenes (5 layouts and 2 wall color permutations). We do the same for the 2-room and 3-room test set. Next, we evenly distribute over the 9 target objects. For instance, for the 0-room test set, we distribute the 216 episodes over 9 target objects (24 per target object).

Table 2: **Target objects selected for each room type.**

Room type	**Target object category**
Kitchen	Fridge
	Kettle
	Apple
Living room	Sofa
	Television
	Newspaper
Bedroom	Bed
	Dresser
	Alarm clock

Appendix C. Additional training details ObjectNav

Table 3 details the hyperparameters we set for all of our training runs. We use DD-PPO [28] and Generalized Advantage Estimation (GAE) [29], parameterized by $\lambda = 0.95$.

Appendix D. Visual representations analysis details

D.1. *Dataset of sampled observations*

For our o.o.d. analysis of agent's visual representations, we generate a small dataset of sampled frames (RGB observations). For this dataset, we sample 200 frames for every room type-wall color combination (3 room types, 3 wall colors) by initializing agents with a random pose in a certain room and sampling its RGB frame. Ground truth data for room type is determined by the agent's position. For instance, if the agent is positioned in the kitchen, the frame encodes a kitchen room type. To ensure the agent is not initialized in e.g., the kitchen but looking towards the bedroom, we limit the possible orientations the agent is initialized at. We limit the orientations such that the agent is not looking into other rooms than the one it is positioned in. This results in 9 sets of frames (200 each), each belonging to a certain room type and wall color combination (e.g. kitchen with red walls). The training set consists of frames showing a bedroom with green walls, a kitchen with red walls and a living room with blue walls. We test on the held-out combinations of room type and wall color, which we split into 'context' vs 'contextless'.

Table 3: **Training hyperparameters.**

Hyperparameter	**Value**
No. of GPUs	2
No. environments per GPU	20
Rollout length	192
No. mini-batches per rollout	1
PPO epochs	4
Discount factor (γ)	0.99
GAE [29] parameter (λ)	0.95
Value loss coefficient	0.5
Entropy loss coefficient	0.01
PPO clip parameter (ϵ)	0.1
Gradient clip norm	0.5
Optimizer	Adam
Learning rate	3e-4

D.2. *Splitting into context/contextless*

This section describes how we split the test set of held-out frames into 'context' vs 'contextless'. First, we cherry pick 6 sampled frames, which we define as ground truth context or contextless. We pick one for each wall color. Next, we extract the 1024-dimensional CLIP embeddings for all frames in our dataset. To form our split, we select the 150 most similar frames to each of the 3 contextless ground truth samples, and 200 most similar frames to each of the 3 context ground truth samples. The similarity is based on cosine score of their encoded CLIP representations. We use the CLIP visual encoder, employing a ResNet-50 backbone.

D.3. *Classifier training details*

For each frame in our dataset, we extract the 1024-dimensional CLIP embeddings using the CLIP visual encoder employing a ResNet-50 backbone. We train a multi-layer perceptron (MLP) to predict room type from these embeddings. We use a MLP with 1 hidden layer (100 neurons wide), ReLu activation, a batch size of 200, a learning rate of 0.0001, an adam optimizer and supervise using the generated ground truth data. Lastly, we standardize individual features using mean removal and scaling to unit variance before training the MLP.

For our analysis of L-B augmented representations (Section 6.4) we first augment the 600 original CLIP embeddings from our train set as detailed in Section 5. This results in 3600 augmented embeddings (6 for each original embedding). For each original embedding, we randomly sample 1 augmented embedding to obtain 600 augmented embeddings for our new train set.

Appendix E. Action space description

Table 4: **Action space description.** We use a 6-action discrete action space.

Action	Description
MOVEAHEAD	Moves the agent forward (if possible) by sampling from $\mathcal{N}(\mu = 0.25m, \sigma = 0.005m)$.
ROTATELEFT ROTATERIGHT	Rotates the agent left or right by sampling from $\mathcal{N}(\mu = 30°, \sigma = 0.5°)$
LOOKUP LOOKDOWN	Tilt the camera of the agent upward or downward by 30°
DONE	Special action of the agent to terminate the episode.

References

[1] A. Khandelwal, L. Weihs, R. Mottaghi and A. Kembhavi, Simple but effective: Clip embeddings for embodied ai, in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE Computer Society, Los Alamitos, CA, USA, jun 2022), pp. 14809–14818.

[2] D. Batra, A. Gokaslan, A. Kembhavi, O. Maksymets, R. Mottaghi, M. Savva, A. Toshev and E. Wijmans, Objectnav revisited: On evaluation of embodied agents navigating to objects, *CoRR* **abs/2006.13171** (2020).

[3] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta and A. Farhadi, AI2-THOR: an interactive 3d environment for visual AI, *CoRR* **abs/1712.05474** (2017).

[4] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh and D. Batra, Habitat: A platform for embodied ai research, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (IEEE Computer Society, Los Alamitos, CA, USA, nov 2019), pp. 9338–9346.

[5] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihs, M. Yatskar and A. Farhadi, Robothor: An open simulation-to-real embodied ai platform, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE Computer Society, Los Alamitos, CA, USA, jun 2020), pp. 3161–3171.

[6] B. Shen, F. Xia, C. Li, R. Martín-Martín, L. Fan, G. Wang, C. Pérez-D'Arpino, S. Buch, S. Srivastava, L. Tchapmi, M. Tchapmi, K. Vainio, J. Wong, L. Fei-Fei and S. Savarese, igibson 1.0: A simulation environment for interactive tasks in large realistic scenes, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Prague, Czech Republic, 2021), pp. 7520–7527.

[7] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge and F. A. Wichmann, Shortcut learning in deep neural networks, *Nature Machine Intelligence* **2**(11) 665–673 (2020).

[8] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba and P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vancouver, BC, Canada, 2017), pp. 23–30.

[9] F. Sadeghi and S. Levine, (cad)$ˆ2$rl: Real single-image flight without a single real image, *CoRR* **abs/1611.04201** (2016).

[10] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, K. Ehsani, J. Salvador, W. Han, E. Kolve, A. Kembhavi and R. Mottaghi, Procthor: Large-scale embodied ai using procedural generation, in *Advances in Neural Information Processing Systems* eds. S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho and A. Oh **35**, (Curran Associates, Inc., 2022), pp. 5982–5994.

[11] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niebner, M. Savva, S. Song, A. Zeng and Y. Zhang, Matterport3d: Learning from rgb-d data in indoor environments, in *2017 International Conference on 3D Vision (3DV)* (Qingdao, China, 2017), pp. 667–676.

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

[13] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu and M. Chen, Hierarchical text-conditional image generation with clip latents, *ArXiv* **abs/2204.06125** (2022).

[14] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen and

I. Sutskever, Zero-shot text-to-image generation, in *Proceedings of the 38th International Conference on Machine Learning* eds. M. Meila and T. Zhang *Proceedings of Machine Learning Research* **139**, (PMLR, 18–24 Jul 2021), pp. 8821–8831.

[15] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger and I. Sutskever, Learning transferable visual models from natural language supervision, in *Proceedings of the 38th International Conference on Machine Learning* eds. M. Meila and T. Zhang *Proceedings of Machine Learning Research* **139**, (PMLR, 18–24 Jul 2021), pp. 8748–8763.

[16] S. Y. Gadre, M. Wortsman, G. Ilharco, L. Schmidt and S. Song, Clip on wheels: Zero-shot object navigation as object localization and exploration, *arXiv preprint arXiv:2203.10421* (2022).

[17] D. Shah, B. Osiński, S. Levine *et al.*, Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action, in *6th Annual Conference on Robot Learning* (Auckland, New Zealand, 2022).

[18] C. Huang, O. Mees, A. Zeng and W. Burgard, Visual language maps for robot navigation, *arXiv preprint arXiv:2210.05714* (2022).

[19] A. Majumdar, G. Aggarwal, B. Devnani, J. Hoffman and D. Batra, Zson: Zero-shot object-goal navigation using multimodal goal embeddings (2022).

[20] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik and S. Savarese, Gibson env: Real-world perception for embodied agents, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT, USA, 2018), pp. 9068–9079.

[21] K. Ehsani, W. Han, A. Herrasti, E. VanderBilt, L. Weihs, E. Kolve, A. Kembhavi and R. Mottaghi, Manipulathor: A framework for visual object manipulation, *CoRR* **abs/2104.11213** (2021).

[22] M. Du, F. He, N. Zou, D. Tao and X. Hu, Shortcut learning of large language models in natural language understanding (2023).

[23] S. Wang, R. Veldhuis, C. Brune and N. Strisciuglio, Frequency shortcut learning in neural networks, in *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications* (New Orleans, USA, 2022).

[24] L. Scimeca, S. J. Oh, S. Chun, M. Poli and S. Yun, Which shortcut cues will dnns choose? A study from the parameter-space perspective, *CoRR* **abs/2110.03095** (2021).

[25] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann and W. Brendel, Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness, *CoRR* **abs/1811.12231** (2018).

[26] D. Gordon, A. Kadian, D. Parikh, J. Hoffman and D. Batra, Splitnet: Sim2sim and task2task transfer for embodied visual navigation, in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019* (IEEE, 2019), pp. 1022–1031.

[27] L. Weihs, J. Salvador, K. Kotar, U. Jain, K. Zeng, R. Mottaghi and A. Kembhavi, Allenact: A framework for embodied AI research, *CoRR* **abs/2008.12760** (2020).

[28] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva and D. Batra, Decentralized distributed PPO: solving pointgoal navigation, *CoRR* **abs/1911.00357** (2019).

[29] J. Schulman, P. Moritz, S. Levine, M. I. Jordan and P. Abbeel, High-dimensional continuous control using generalized advantage estimation, in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* eds. Y. Bengio and Y. LeCun (San Juan, Puerto Rico, 2016).