Supplementary for "TIDEE: Tidying Up Novel Rooms using Visuo-Semantic Commonsense Priors"

Gabriel Sarch[1]*, Zhaoyuan Fang[1], Adam W. Harley[1], Paul Schydlo[1], Michael J. Tarr[1], Saurabh Gupta[2], and Katerina Fragkiadaki[1]

[1] Carnegie Mellon University
[2] University of Illinois at Urbana-Champaign

*Correspondence to gsarch@andrew.cmu.edu

1 Overview

Section 2 contains more details of the methods described in the main paper. Section 3 provides additional details on the experiments. Section 4 provides additional evaluation of the networks.

2 Implementation details

2.1 Virtual environment and action space

We use the following actions: move forward, rotate right, rotate left, look up, look down, pick up, put down. We rotate in the yaw direction by 90 degrees, and rotate in the pitch direction by 30 degrees. We do not constrain our agent to grid locations. The RGB and depth sensors are at a resolution of 480x480, a field of view of 90 degrees, and lie at a height of 0.9015 meters. The agent's coordinates are parameterized by a single (x, y, z) coordinate triplet with x and z corresponding to movement in the horizontal plane and y reserved for the vertical direction. Picking up objects occurs by specifying an (x,y) coordinate in the agent's egocentric frame. If by ray-tracing, the point intersects an object that is pickupable and within 1.5 meters of the agent, then the pickup action succeeds. Placing objects occurs by specifying an (x,y) coordinate in the agent's egocentric frame to place the object. If by ray-tracing, the point intersects an object that is a receptacle class, has enough free space in the radius of the target location, and within 1.5 meters of the agent, then the place action succeeds if the agent is holding an object. Since some objects require their state to be open for placement to successfully occur (e.g. Fridge), the agent will also try to open the receptacle if placement initially fails.

2.2 Pseudo code for TIDEE

We present pseudo code for the TIDEE algorithm in Algorithm 1. We denote FMM to mean Fast Marching Method [6], g to denote the point goal in the 2D overhead map $\mathbf{M}^{2\mathrm{D}}$, r to denote a receptacle, and fps to denote farthest point sampling. If TIDEE does not find one of the predicted receptacles from the rGCN network, TIDEE will attempt to retrieve a general receptacle class from its memory of detected objects, navigate there, and attempt to place it. If after m placement attempts the object is still not placed successfully (for example if TIDEE gets stuck while navigating), TIDEE will drop the object at its current location and resume the out-of-place search.

Algorithm 1 TIDEE algorithm

```
while $unexplored\_area > A$ do                              ▷ Mapping the scene
    if g reached then
        Sample new g in unexplored area
    end if
    Execute movement with FMM to g
    Update $\mathbf{M}^{2\mathrm{D}}$, $\mathbf{M}^{3\mathrm{D}}$, $\mathcal{M}^{\mathrm{O}}$
end while
Sample new g in reachable area                          ▷ out-of-place detection
while not $oop$ found after sampling $k$ goals do
    if g reached then
        Sample new g in reachable area
    end if
    Execute movement with FMM to g
    Update $\mathbf{M}^{2\mathrm{D}}$, $\mathbf{M}^{3\mathrm{D}}$, $\mathcal{M}^{\mathrm{O}}$
    Run dDETR+BERT-OOP
    if oop found then
        navigate to oop, Execute PickupObject
        $r \leftarrow$ Run rGCN                          ▷ Infer plausible context
        if $r \in \mathcal{M}^{\mathrm{O}}$ then
            navigate to $r$ with FMM, Execute PutObject
        else
            $m \leftarrow$ Run $f_{\text{search}}$                     ▷ Localize context
            for g $\in$ fps($m$) do
                navigate to g with FMM
                if $r$ detected then
                    navigate to $r$ with FMM
                    Execute PutObject
                end if
            end for
        end if
    end if
end while
```

2.3 Semantic mapping and planning

TIDEE maintains two spatial visual maps of its environment that it updates at each time step from the input RGB-D stream: i) a 2D overhead occupancy map

$\mathbf{M}_t^{2\mathrm{D}} \in \mathbb{R}^{H \times W}$ and, ii) a 3D occupancy and semantics map $\mathbf{M}_t^{3\mathrm{D}} \in \mathbb{R}^{H \times W \times D \times K}$, where K is the number of semantic object categories, we use $K = 116$. The $\mathbf{M}^{2\mathrm{D}}$ maps are used for exploration and navigation in the environment. The $\mathbf{M}^{3\mathrm{D}}$ maps are used for inferring locations of potential receptacles conditioned on their semantic categories, as described in Section 3.4 of the main paper.

At every time step t, we unproject the input depth maps using intrinsic and extrinsic information of the camera to obtain a 3D occupancy map registered to the coordinate frame of the agent, similar to earlier navigation agents [1]. The 2D overhead maps $\mathbf{M}_t^{2\mathrm{D}}$ of obstacles and free space are computed by projecting the 3D occupancy along the height direction at two height levels and summing. For each input RGB image, we run a state-of-the-art d-DETR detector [9] (pretrained on COCO [4] then finetuned on AI2THOR) to localize each of K semantic object categories. Similarly, we use the depth input to map detected 2D object bounding boxes into a 3D centroids dilated with Gaussian filtering and add them into the 3D semantic map, we have one channel per semantic class—similar to [2], but in 3D as opposed to a 2D overhead map. We did not use 3D object detectors directly because we found that 2D object detectors are more reliable than 3D ones simply because of the tremendous pretraining in large-scale 2D object detection datasets, such as MS-COCO [4]. Finally, 3D maps $\mathbf{M}^{3\mathrm{D}}$ result from the concatenation of the 3D occupancy maps with the 3D semantic maps. Alongside the 3D semantic map $\mathbf{M}^{3\mathrm{D}}$, we maintain an object memory $\mathcal{M}^{\mathrm{O}}$ as a list of object detection 3D centroids and their predicted semantic labels $\mathcal{M}^{\mathrm{O}} = \{[(X, Y, Z)_i, \ell_i \in \{1...K\}], i = 1..K\}$, where K is the number of objects detected thus far. The object centroids are expressed with respect to the coordinate system of the agent, and, similar to the semantic maps, updated over time using egomotion.

Exploration and path planning TIDEE explores the scene using a classical mapping method. We take the initial position of the agent to be the center coordinate in the map. We rotate the agent in-place and use the observations to instantiate an initial map. Second, the agent incrementally completes the maps by randomly sampling an unexplored, traversible location based on the 2D occupancy map built so far, and then navigates to the sampled location, accumulating the new information into the maps at each time step. The number of observations collected at each point in the 2D occupancy map is thresholded to determine whether a given map location is explored or not. Unexplored positions are sampled until the environment has been fully explored, meaning that the number of unexplored points is fewer than a predefined threshold.

To navigate to a goal location, we compute the geodesic distance to the goal from all map locations using a fast-marching method [6] given the top-down occupancy map $\mathbf{M}^{2\mathrm{D}}$ and the goal location in the map. We then simulate action sequences and greedily take the action sequence which results in the largest reduction in geodesic distance.

2.4 2D-to-3D unprojection

For the i-th view, a 2D pixel coordinate (u, v) with depth z is unprojected and transformed to its coordinate $(X, Y, Z)^T$ in the reference frame:

$$(X, Y, Z, 1) = \mathbf{G}_i^{-1} \left(z \frac{u - c_x}{f_x}, z \frac{v - c_y}{f_y}, z, 1 \right)^T \tag{1}$$

where (f_x, f_y) and (c_x, c_y) are the focal lengths and center of the pinhole camera model and $\mathbf{G}_i \in SE(3)$ is the camera pose for view i relative to the reference view. This module unprojects each depth image $I_i \in \mathbb{R}^{H \times W \times 3}$ into a pointcloud in the reference frame $P_i \in \mathbb{R}^{M_i \times 3}$ with M_i being the number of pixels with an associated depth value.

We voxelize the point cloud into a 128x64x128 occupancy $\in \{0, 1\}$ centered at the initial position of the agent, and aggregate (take max) the occupancies across views to obtain $M_t^o \in \{0, 1\}$.

2.5 Object tracking and semantic aggregation.

As described in Section 3.2, we track previously detected objects by their 3D centroid $C \in \mathbb{R}^3$. We estimate the centroid by taking the 3D point corresponding to the median depth within the bounding box detection and bring it to a common coordinate frame. We extend previous work [2] to 3D and add a channel to the 3D occupancy map for each object category. For each detected centroid C^j of class index j, we accumulate it into a 3D occupancy map. We then apply a Guassian filter g to dilate the centroids in the map and add this to to the jth channel of the 3D semantic occupancy map M_t. Thus, the jth channel of the 3D semantic map at time step t can be written as:

$$M_t^j = M_t^o + g(f(C^j)) \tag{2}$$

where $M_t^o \in \mathbb{R}^{H \times W \times D}$ is the accumulated 3D occupancy, g is a guassian filter operation, and f accumulates each centroid i in class index j into an occupancy map $M \in \mathbb{R}^{H \times W \times D}$. Centroids are more robust to noisy depth and detection estimates, and often provide enough information for active search and object spatial tracking.

2.6 Out-of-place detector

As described in Section 3.2 of the main paper, our OOP detector makes use of visual and relational language as input to our OOP network. We generate training scenes with some objects out-of-place using the same algorithm described in Section 3.1. We first finetune deformable-DETR [9] (pretrained on COCO [4]) on the training houses (object seed randomized) to predict the bounding boxes, semantic segmentation masks, and semantic labels by generating random trajectories through the scene. We then train on the messup configurations and add an additional classification loss on the output decoder queries to predict whether the object is in- or out-of-place. We use the output decoder queries for the `dDETR-OOP` classifier.

For the language detector, we freeze the detector described above, and use it to update our object tracker $\mathcal{M}^{\mathrm{O}}$ while the agent explores the scene. Then, the agent visits a location to search for an out-of-place object and for each object detected in view above a confidence threshold, we infer its relations described in Section 2.7 with all objects in memory, and systematically combine them into a paragraph of text. An example paragraph is shown below. *The pillow is next to the key chain. The pillow is next to the laptop. The pillow is next to the side table. The pillow is next to the mug. The pillow is next to the teddy bear. The pillow is supported by the side table. The pillow is closest to the mug.* We make use of the extensive pretraining of the BERT language model [3] as a starting point for our language classifier. We tokenize the paragraph text and give it as input to the BERT model. For the language-only detector (`BERT-OOP`), we give the pooled output {cls} token from BERT to a three-layer fully-connected classifier to predict in or out-of-place.

For the language and visual detector (`dDETR+BERT-OOP`), we concatenate the pooled output {cls} token from BERT with the output query embedding corresponding to the detected object from deformable-DETR, and give this concatenated embedding to a three-layer fully-connected classifier to predict in or out-of-place. We train the classifiers using known labels of in or out-of-place from our mess up algorithm.

For the BERT-only model, we give the pooled output {cls} token from BERT as input to our classifier. For the visual-only model, we give the output query embedding corresponding to the detected object from deformable-DETR to the classifier.

We use the same hyperparameters for training all classifiers. We use a batch size of 25, an AdamW optimizer with a learning rate of 2e-7 and weight decay of 0.01, and train for 20k iterations.

2.7 Object centroid relations

As described in Section 3.2 of the main paper, we define a set of three relations based on the estimated centroids of the detected objects within the scene. We use these relations for building our input to the BERT out-of-place detector. These relations are computed with the following metrics:

(i) *Supported-by*: A receptacle is defined as a type of object that can contain or support other objects. Sinks, refrigerators, cabinets, and tabletops are some examples of receptacles. For the floor receptacle class, we consider the point directly below the object at the height of the floor (lowest height in our map). For all centroids C_t^{rec} corresponding to receptacle classes $L_t^{\mathrm{rec}} \subseteq L_t$, we define the single object $L^{\mathrm{supp}} \in L_t^{\mathrm{rec}}$ that supports the detected C^{det} object as:

$$L^{supp} = \arg\min(D(C^{\mathrm{det}}, C^{\mathrm{rec}}_{t;\mathrm{ydiff}<0})) \qquad (3)$$

Where $D(x, Y)$ is the euclidean distance between centroid x and each centroid in Y, and ydiff < 0 takes all tracked centroids which are below the height of the detected centroid.

(ii) *next-to*: We define the objects L^{next} that are next to the detected C^{det} object as:

$$L^{next} = D(C^{det}, C_t) < d \tag{4}$$

Where $D(x, Y)$ is the euclidean distance between centroid x and all centroids Y, and d is a distance threshold.

(ii) *closest-to*: We define the single object L^{closest} that is closest to the detected C^{det} object as:

$$L^{\text{closest}} = \arg\min(D(C^{\text{det}}, C_t)) \tag{5}$$

Where $D(x, Y)$ is the euclidean distance between centroid x and all centroids Y.

2.8 Relational graph convolutional network

As described in Section 3.3 of the main paper, we use a relational graph convolutional network to predict plausible receptacle classes for the out-of-place object. The memex graph nodes are the sum of a learned object category embedding and visual features obtained from cropping the deformable-DETR backbone with the object's bounding box at the closest navigable location to the object. We connect nodes in the memory graph by computing their relations as described in Section 2.9. For the out-of-place object node, we similarly sum the learned embedding of the object's category label and visual features obtained from cropping the deformable-DETR backbone with the detected bounding box. The scene graph nodes are deformable-DETR output query features in the initial mapping of the scene for all detections above a confidence threshold. We include a map type node which is initialized with a learned embedding for each of the four room types.

We use the rGCN to message pass 1) within the memory graph, and 2) to bridge the memory, scene, and out-of-place nodes. Let n_{OOP} denote the node of the out-of-place object initialized with a learned category class embedding and visual features.

Following the rGCN formulation in [5], we first update the nodes in the memory graph to distribute information within the memory:

$$h_i^{(l+1)} = \sigma\Big(\sum_{r \in \mathcal{R}^{mem}} \sum_{j \in \mathcal{N}_{i,r}^{mem}} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\Big), \tag{6}$$

where $h_i^{(l)} \in \mathbb{R}^{d^{(l)}}$ is the hidden state of node v_i in the l-th layer of the neural network, with $d^{(l)}$ being the dimensionality of this layer's representations, $\mathcal{N}_{i,r}^{mem}$ denotes the set of memory neighbor indices of node i under relation $r \in \mathcal{R}^{mem}$, and $c_{i,r}$ is a problem-specific normalization constant.

Inspired by [8], we then define a set of four bridging edges $\mathcal{R}^{bridge}$, one to connect n_{OOP} to the updated memory nodes of the same object class, one to connect n_{OOP} to all current scene nodes, one to connect n_{OOP} to the room type node, and one to connect the the updated memory nodes to current scene nodes with the same category label. We then message pass via the bridging edges:

$$h_i^{(l+1)} = \sigma\Big(\sum_{r \in \mathcal{R}^{bridge}} \sum_{j \in \mathcal{N}_{i,r}^{bridge}} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\Big), \tag{7}$$

where $\mathcal{N}_{i,r}^{bridge}$ denotes the set of bridge neighbor indices of the target node under bridge relation $r \in \mathcal{R}^{bridge}$.

We use four relational graph convolutional layers for each stage of message passing. Finally, we run the updated out-of-place object node through a classifier layer to predict a probability distribution over proposed receptacle classes to search for placing the target object. We optimize with a cross entropy loss using the object's ground truth receptacle label from the training scenes.

2.9 Memex graph

We use 20 of the 80 training rooms to construct the memex graph. As described in section 3.3 of the main paper, the memex graph is a large graph of object nodes and relational edges that provide the relational graph convolutional network with exemplar context of object-object and object-scene relations. We obtain the ground truth category labels for the objects and use ground truth information from the simulator to obtain the relations *above*, *below*, *next to*, *supported by*, *aligned with*, and *facing*. The memex remains a constant graph throughout all remaining training and testing scenes. We use simulator ground truth information for convenience, but note that we could instead obtain the neural memex graph from human annotations of real-world houses. We compute *above*, *below*, *next to*, and *supported by* similar to Section 2.7, but instead use a distance metric on the 3D bounding boxes. For *aligned with*, we check if the 3D bounding boxes have parallel faces. For *facing*, we note that the back of an object usually carries more of its mass (e.g. the back of a sofa). Thus, we look at the mass distribution of the object within its 3D bounding box, and take the box face with the most of the point mass in its direction to be the back of the object. An object is facing a second object if the frustum of its front 3D bounding box face intersects the second object. We only consider facing for the following classes: *Toilet, Laptop, Chair, Desk, Television, ArmChair, Sofa, Microwave, CoffeeMachine, Fridge, Toaster*.

2.10 Visual search network

As described in Section 3.4 of the main paper, we use a visual search network to propose search locations conditioned on an object class. The input to the network is a 3D occupancy map $\in \mathbb{R}^{C \times D \times H \times W}$ with $C = 116$, $D = 64$, $H = 128$, $W = 128$. $C = 116$ represents a channel for each possible category in AI2THOR, as described in Section 2.5. We first tile classes along all heights in $\mathbf{M}^{\mathrm{3D}}$ to obtain a 2D input $\in \mathbb{R}^{(C \cdot D) \times H \times W}$ to the network. This enters four

2D convolutional layers and returns a feature map $V^{uncond} \in \mathbb{R}^{C \times H \times W}$. The target object class is encoded with a learned category embedding and matrix multiplied with the feature map to condition the network on the target class. This is sent as input to four additional 2D convolutional layers to get a final output map $V^{cond} \in \mathbb{R}^{H \times W}$. We optimize this with a binary cross entropy loss on each 2D position independently using a Guassian-smoothed 2D map of ground truth object positions in the training scenes. Our output map provides spatial positions at a resolution of 128×128. Since our output map need not predict a single location to search, we give positive samples significantly larger class weight than the negative samples to encourage high recall of the true location in the thresholded area.

3 Experimental details

3.1 Tidying task

Our tidying task begins with moving N objects out of their natural locations in the scene. We use $N = 5$ and generate five messy configurations per test room (total of 20 rooms × 5 configurations = 100 test configurations). For each object to be moved out-of-place, we randomly select a pickupable object, spawn an agent to a random navigable location in the scene at a random orientation in increments of 90 degrees, and with probability p, drop the object at the agent's location, or with probability $1 - p$, throw the object with a constant force and let AI2THOR's physics engine resolve the final location (action "ThrowObject" in AI2THOR). We use $p = 0.5$. In AI2THOR, the throw distance of an object depends on its pre-defined mass, and thus the throw distance will change depending on the object. We keep the throw force constant at 150.0 newtons. We disable object breaking so that no objects are changed to their breaking state after dropping or throwing them. We show examples of out-of-place objects in Figure 1.

We define an episode as the time from the spawn of the agent in the messy environment to the time the agent executes the "done" action, or 1000 steps have been taken (whichever comes first). Once the tidying episode begins, the agent is spawned near the center of the map. At each time step, the agent is given an RGB and depth sensor, and its exact egomotion in terms of how far each action takes the agent and in what direction. During the out-of-place detection phase, TIDEE samples random locations within its 2D map to search.

3.2 Human placement evaluation

We report in Section 4.2 of the main paper a human evaluation of TIDEE placements compared to baselines. We use the Amazon Mechanical Turk interface to query human evaluators as to whether they prefer TIDEE placements compared to baseline placements. For all successful placements by the agents, we generate three images of each placement to show the object from three distinct viewing

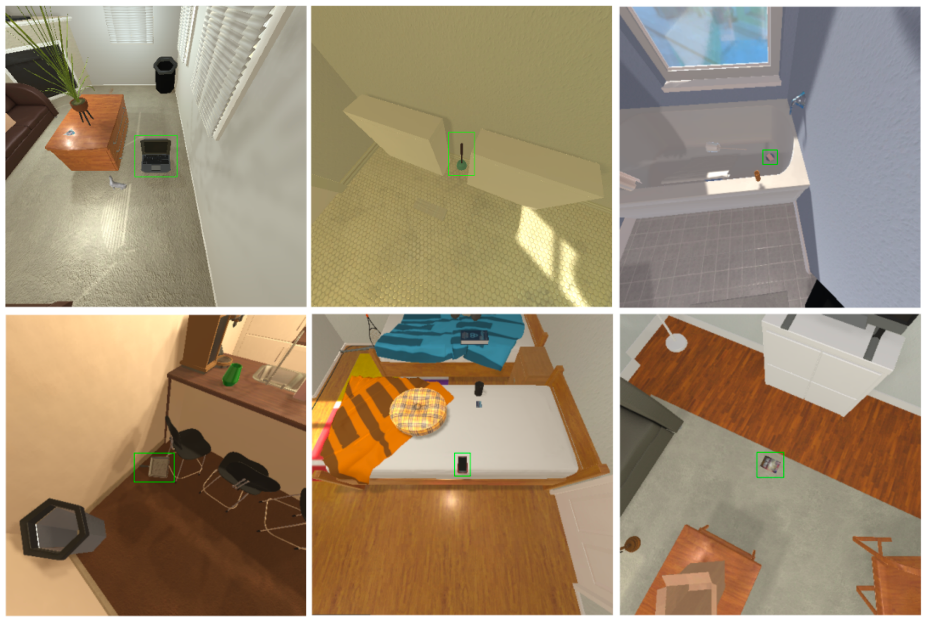

Fig. 1. Example images of out-of-place objects.

angles, as shown in Figure 2. We instruct the evaluators to choose between the placements of TIDEE and the baseline placement by looking at the images and picking which position of the object they would prefer. The full instructions given to the human evaluators for an example statue placement is displayed below. For this evaluation, we only consider objects which were picked up by both agents (TIDEE and the baseline).

Consider a scenario where you are putting the statue into its correct location in a room. Please choose which location you would prefer to place the statue within the room. The two options (A & B) represent two different possible locations of the statue in the same room (in the images the location of the statue is shown with a box). Each option (A & B) show the object from three distinct camera angles to help you make your decision. Important: Please judge only by the placement location of the object within the room, and NOT by the orientation of the object on the supporting surface.

3.3 Out-of-place detection evaluation

We evaluate the out-of-place detector performance in Section 4.3 on the same messy test scenes used for the tidying-up task. We generate 20 random views of each messy configuration where at least one out-of-place objects is in view. The total evaluation consists of 2000 images (20 scenes $\times$ 5 configurations $\times$ 20 views = 2000). We evaluate each detector by measuring average precision across all the images, where in and out-of-place are the two categories.

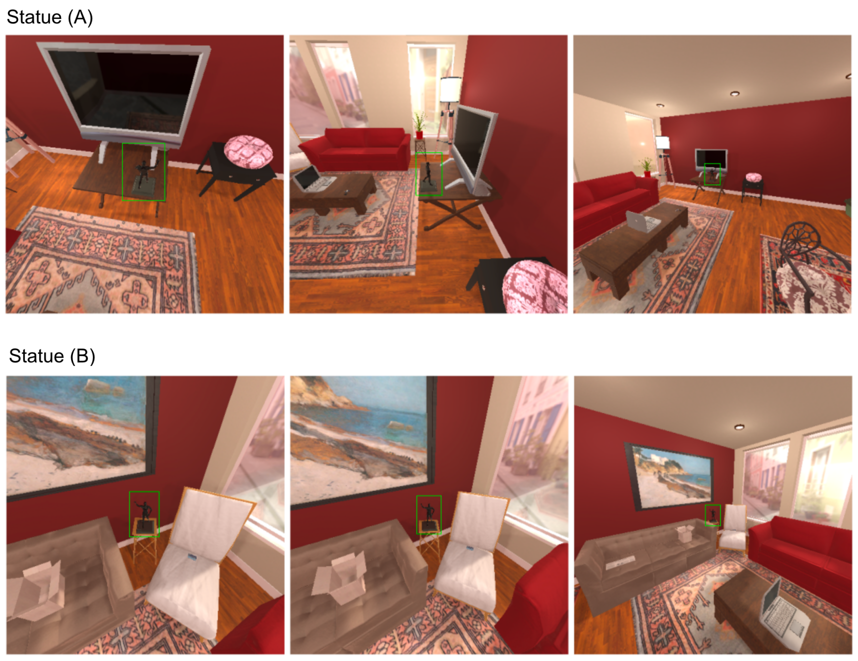

Fig. 2. Example images shown to Amazon Mechanical Turk evaluators.

3.4 Exploration with visual search network evaluation

We evaluate the visual search network to assist in object goal navigation for objects in their default locations in the AI2THOR test scenes (20 scenes in total) in Section 4.4. For each test scene, the agent is tasked with finding each object category that exists at least once in the test scene. Each episode involves finding an instance of a given category. We consider all object categories across the AI2THOR simulator (116 categories). Tasking the agent under these specifications provides 591 total episodes in the evaluation. As mentioned in the main text, the agent is successful when the agent is within 1.5 meters of the target object and the object is visible to the agent. To declare success, the agent must execute the ”Stop” command. If ”Stop” is not executed within the maximum number of steps (200 max), the episode is automatically considered a failure and the next episode will begin. Both TIDEE and the baseline presented in Table 2 of the main text use the same object detector and navigation modules from Section 3.1 of the main paper. The only difference is how the model selects locations in the scene to search for the object-of-interest. For both TIDEE and the baseline, the agent executes the ”Stop” command after the object category has been detected above a threshold and the agent has navigated to the detected object using the estimated 3D centroid.

3.5 Updating placement priors by instruction

We show that we can alter the output of the language out-of-place detector by pairing specific language input with a desired label after additional training in Section 4.3. To do so, we first train the language detector (`BERT-OOP`) as described in Section 2.6 and Section 3.2 of the main paper. We then target a relation-label pairing. For example, we may want the relation "alarm clock supported-by the desk" to output the label "out-of-place" (which does not appear in the unaltered training set) whenever the relation occurs. Then, for an additional amount of (9k) iterations, whenever the relation "alarm clock supported-by the desk" appears in the training batch, we pair the sample with the "out-of-place" label as supervision.

4 Additional results

4.1 2021 Rearrangement Challenge

In section 4.5 of the main paper, we report the performance of TIDEE on the 2022 rearrangement benchmark. We additionally report performance on the 2021 rearrangement benchmark in Table 1.

Table 1. Test set performance on 2-Phase Rearrangement Challenge (2021).

	% FixedStrict ↑	% Success ↑	% Energy ↓	% Misplaced ↓
TIDEE	**8.9**	**2.6**	**93**	**95**
TIDEE *+noisy pose*	6.6	1.9	97	98
TIDEE *+est. depth*	5.5	1.4	96	97
TIDEE *+noisy depth*	8.9	2.3	93	95
Weihs *et al.* [7]	1.4	0.3	110	110

4.2 Visualizations of the Visual Search Network

In Section 4.4 of the main paper, we displayed visualizations of the Visual Search Network predictions. We provide additional visualizations of the sigmoid output of our Visual Search Network conditioned on an object category in test rooms in Figure 3. We display an overhead view of the full scene on the left, and the network predictions corresponding to the overhead spatial locations on the right conditioned on four randomly-selected object categories. Darker red corresponds to higher probability. The blue dot indicators plotted in the prediction maps correspond to the search locations for the agent to visit after thresholding and farthest point sampling (for # location = 3). The output generally puts the highest probability at plausible areas for the category to exist. However, occasionally the network puts high probability where it should not. For example,

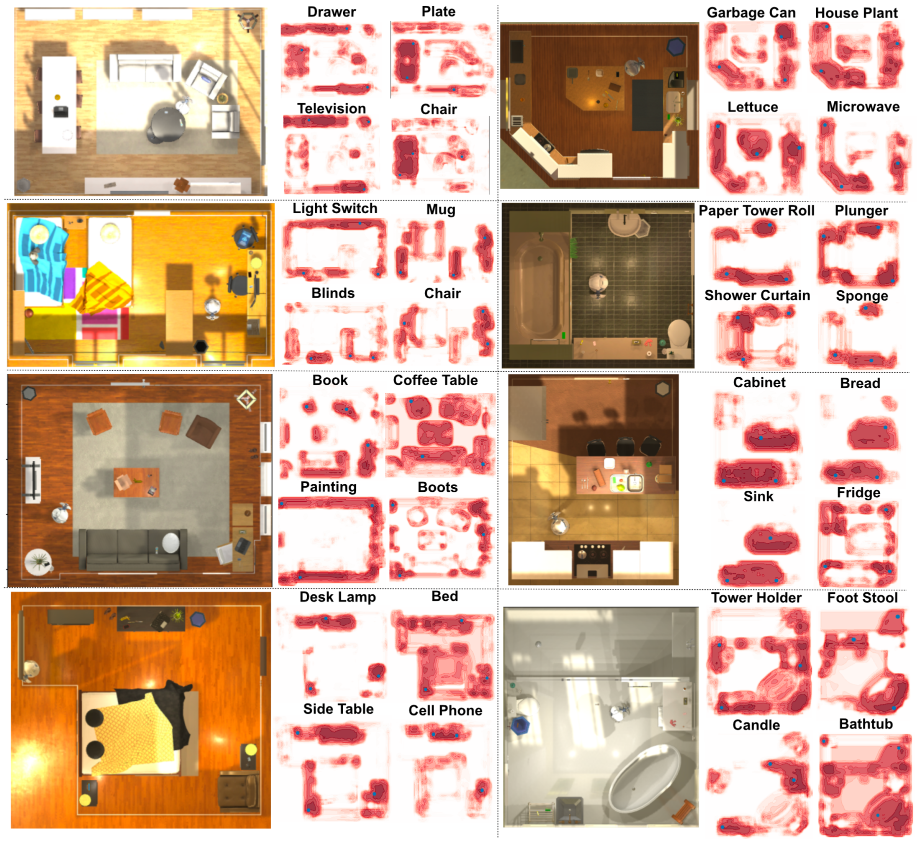

Fig. 3. Examples of the output of the Visual Search Network in test scenes.

the network puts high probability near a dresser for category "Bed", or near the armchair for category "Coffee Table". This may be in part due to our training procedure to prioritize high recall over precision of the true location in our cross entropy weighting.

4.3 Evaluation of altering priors with natural language

In Section 4.6 of the main paper, we showed for a single example that we can alter the learned priors of the out-of-place detector using external language input. We augment training with nine additional object relation pairs that are among the most commonly found in the AI2THOR houses and pair the relation with an out-of-place label. The relation pairs include "alarm clock is supported by desk" (from main text), "Soap bottle is supported by countertop", "Pen is supported by desk", "Laptop is supported by desk", "Pillow is supported by bed", "Toilet paper is support by toilet", "salt shaker is supported by countertop", "Spatula is supported by countertop", "Statue is supported by shelf", and "Vase is supported by shelf". We follow the same training procedure as in Section 3.5. The average change in probability across test houses for examples where the relation appears is shown in Table 2. The significant change in probability indicates we are able to change the detector output with simple language instructions.

Table 2. Altering priors with instructions. The out-of-place confidence of the out-of-place classifier before and after augmenting training with the uncommon relation-label pairing.

	Before instruction	After instruction
Alarm Clock supported-by Desk	.10	.70
Knife supported-by Dining Table	.44	.91
Bowl supported-by Dining Table	.23	.71
SoapBar supported-by Toilet	.21	.68
Laptop supported-by Bed	.25	.71
Apple supported-by CounterTop	.14	.62
Mug supported-by CounterTop	.27	.77
Newspaper supported-by Sofa	.43	.98
Pillow supported-by Bed	.56	.70
Book supported-by Desk	.63	.88

References

1. Chaplot, D.S., Gandhi, D., Gupta, S., Gupta, A., Salakhutdinov, R.: Learning to explore using active neural slam. In: International Conference on Learning Representations (ICLR) (2020) 3
2. Chaplot, D.S., Jiang, H., Gupta, S., Gupta, A.: Semantic curiosity for active visual learning. In: European Conference on Computer Vision. pp. 309–326. Springer (2020) 3, 4
3. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019). https://doi.org/10.18653/v1/N19-1423, https://aclanthology.org/N19-1423 5
4. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014) 3, 4
5. Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European semantic web conference. pp. 593–607. Springer (2018) 6
6. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences **93**(4), 1591–1595 (1996) 1, 3
7. Weihs, L., Deitke, M., Kembhavi, A., Mottaghi, R.: Visual room rearrangement. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2021) 11
8. Zareian, A., Karaman, S., Chang, S.F.: Bridging knowledge graphs to generate scene graphs. In: European Conference on Computer Vision. pp. 606–623. Springer (2020) 6
9. Zhu, X., Su, W., Lu, L., Li, B., Wang, X., Dai, J.: Deformable {detr}: Deformable transformers for end-to-end object detection. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=gZ9hCDWe6ke 3, 4