Pushing it out of the Way: Interactive Visual Navigation

Kuo-Hao Zeng[1] Luca Weihs[2] Ali Farhadi[1] Roozbeh Mottaghi[1,2]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]PRIOR @ Allen Institute for AI
prior.allenai.org/projects/interactive-visual-navigation

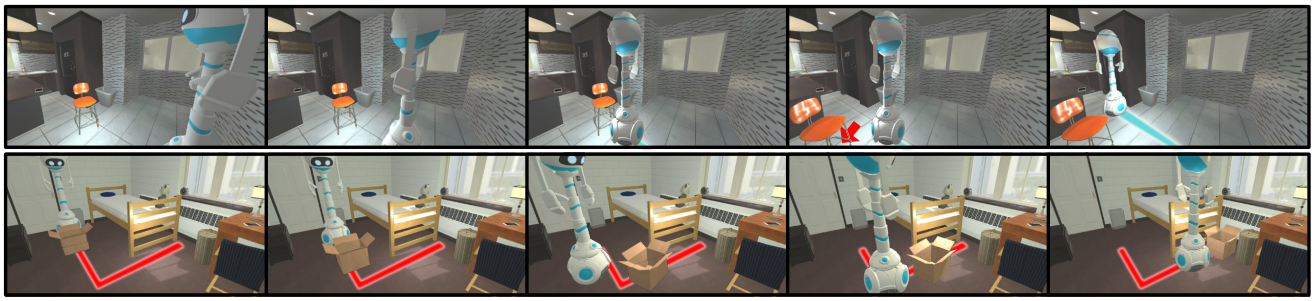

Figure 1: Visual navigation may require interactions that go beyond moving forward/backward, and turning left/right. For example, the agent in the top row needs to push the chair out of its way to reach the target. Interactive navigation entails deeper understanding of the outcome of agents actions on objects in the scene. In this paper, we introduce Neural Interaction Engine (NIE) to explicitly predict the effect of actions on objects poses. By integrating NIE with our policy network we show that we can perform long-horizon planning while predicting the outcome of the actions. We evaluate NIE for visual navigation where the path to the goal is obstructed, and moving objects to specific locations in the scene and show major improvements over state of the art in these tasks.

Abstract

We have observed significant progress in visual navigation for embodied agents. A common assumption in studying visual navigation is that the environments are static; this is a limiting assumption. Intelligent navigation may involve interacting with the environment beyond just moving forward/backward and turning left/right. Sometimes, the best way to navigate is to push something out of the way. In this paper, we study the problem of interactive navigation where agents learn to change the environment to navigate more efficiently to their goals. To this end, we introduce the Neural Interaction Engine (NIE) to explicitly predict the change in the environment caused by the agent's actions. By modeling the changes while planning, we find that agents exhibit significant improvements in their navigational capabilities. More specifically, we consider two downstream tasks in the physics-enabled, visually rich, AI2-THOR environment: (1) reaching a target while the path to the target is blocked (2) moving an object to a target location by pushing it. For both tasks, agents equipped with an NIE significantly outperform agents without the understanding of the effect of the actions indicating the benefits of our approach.

1. Introduction

Embodied AI has witnessed remarkable progress over the past few years owing to advances in learning algorithms, benchmarks, and standardized tasks. A popular task that has received a considerable amount of attention is visual navigation [3, 5, 8, 29, 39, 48], where the goal is to navigate towards a specific coordinate or object within an unseen environment. One of the common implicit assumptions for these navigation methods is that the scene is static, and the agent cannot interact with the objects to change their pose.

Consider the scenario that the path of the agent towards the target location is blocked by an obstacle (e.g., a chair) as shown in Fig. 1 (top). To reach the target, the agent has to move the obstacle out of the way. Therefore, planning for reaching the target requires not only understanding the outcome of agent actions but also the dynamics of agent-object interactions. There are many factors such as object size, spatial relationship with other objects in the scene, and

reaction of the object to the applied forces, that influence the outcome of the interaction with the object. Hence, long-horizon planning for navigation conditioned on the object dynamics offers unique challenges that are often overlooked in the recent navigation literature.

The first challenge is to learn whether an action affects the pose of an object or not. Navigation actions (e.g., rotate right or move ahead) typically do not affect the position of objects in the world coordinate frame while interaction actions (e.g., pushing an object) can change the object pose. The objects move in the ego-centric view of the agent due to agent movements or interaction with objects. Learning how objects move as a result of camera motion or interaction imposes the second challenge. Learning how to interact with objects is another challenge. For example, the agent should learn that pushing an object against a wall does not change its pose.

In this paper, we propose a novel model for navigation while interacting with objects within a scene that jointly plans a sequence of actions and predicts the changes in the scene conditioned on those actions. More specifically, the model includes a Neural Interaction Engine (NIE) module that predicts the affine transformation of objects from the perspective of the agent conditioned on the actions. The goal is to learn if/how the actions affect the pose of the objects. The NIE module receives gradients for not only the prediction of the pose in the next frame but also the navigation policy.

We evaluate our model on two downstream tasks *ObsNav* and *ObjPlace*. The goal of *ObsNav* is to reach a specific coordinates in a scene while the paths from the initial location of the agent to the target are blocked by objects. The goal of *ObjPlace* is to push an object on the floor while navigating so it reaches a target point. These are challenging tasks since the agent requires an accurate understanding of the dynamics of the objects and their interaction with other objects in the scene. We perform our experiments in 120 scenes of the physics-enabled AI2-THOR [19] environment. Our experiments show significant improvement over baselines that are not capable of explicitly predicting the effect of interactions showing the merit of our NIE model.

In summary, we highlight three primary contributions. (1) We propose Neural Interaction Engine, as a model for predicting the state of the observed objects conditioned on the agent actions. (2) We propose new datasets for two navigation-based tasks using a physics-enabled framework, which enables changing the pose of objects and models rich object-object and agent-object interactions. (3) We show that predicting the outcome of actions is a crucial capability for embodied agents by showing significant improvements over baselines that do not possess this capability.

2. Related Work

Action-conditioned learning of rigid body dynamics. The goal of these works is to learn the dynamics of rigid body motion under the effect of applied actions. Byravan and Fox [6] segment a point cloud into salient regions and predict the rigid body motion. Li *et al*. [21] learn to re-position and re-orient an object with unknown physical properties. Several works [11, 12, 13, 46] have proposed formulations of visual Model Predictive Control, where the central insight is that a predictive model of sensory input is a powerful signal for learning to perform tasks. A number of other strategies for action-conditioned learning have been proposed, these include: learning latent physical properties of objects using visual observation of interactions with those objects [45], learning forward and inverse scene dynamics from object interaction data [26], representing scenes as object-centric graphs and learning to predict changes in object pose after applying a push action [28], learning the dynamics of balls and walls in the game of billiards [14], and modeling the dynamics of robot interactions by jointly estimating forward and inverse models of dynamics [1]. In contrast to all of these approaches, we consider the more complex mobile robot scenario, where we factorize the effect of robot motion and object motion.

Learning dynamics from perception. The dynamics of objects can be inferred from images and videos alone without any interaction. [35] decompose frame-to-frame pixel motion into scene depth, 3D camera rotation and translation, and a set of object regions with their corresponding 3D motion. [22] reason about the underlying physical properties of objects that appear in a sequence of frames and predict future motion of those objects. [17, 41] jointly train a perception module, an object-based physics engine and a renderer to generate the future predictions. [7] propose Neural Physics Engine that outputs the future states of objects and their properties. [36] also infers the physical state of objects from video input and predict their future trajectories. [40] infer physical properties of objects such as mass and density from videos. [24, 25, 46, 47] predict the dynamics of objects and their future trajectory. These approaches focus on simple scenarios (such as balls of uniform mass or a stack of cubes), no agent action is considered or assume a static camera.

Visual navigation. The tasks that we consider in this paper involves visual navigation. Visual navigation has been addressed in various papers in recent Embodied AI literature. Most works focus on point navigation (PointNav) [3, 9, 29, 38] or object navigation (ObjectNav) [5, 8, 10, 39]. Our task is different since in these works only static scenes are considered.

Our task is closer to existing tasks that consider navigation among movable obstacles [4, 18, 23, 31, 32, 43, 44]. The difference with [31, 32] is that those works are not

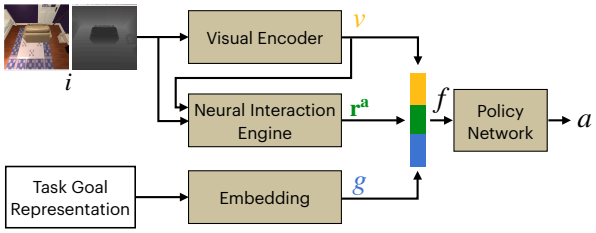

Figure 2: **Model overview.** Our model includes three main parts: Visual Encoder, Neural Interaction Engine, and Policy Network.

learning-based and generalization to unseen scenes is not evaluated. Our task differs from that of [44] in that our agent applies forces to objects with different magnitudes and directions (as opposed to moving objects by colliding with them). Our approach also shows significant improvements over the vanilla RL approaches used in [44].

3. Model

In this section, we begin by providing an overview of the proposed model. We then introduce our Neural Interaction Engine (NIE) and explain how we integrate the NIE into the policy network. Finally, we describe the learning objective and how we learn the entire model with the NIE module.

3.1. Model Overview

Our model has three main components: a visual encoder, Neural Interaction Engine, and a policy network, as illustrated in Fig. 2. First, the visual encoder produces a representation v from a visual observation i. The visual observation includes an RGB image captured by a mounted camera and a depth image captured by a depth sensor. The visual encoder is a convolutional neural network aiming to extract informative features from the given observation. Second, the NIE, which receives the same input observation i, extracts keypoints $\mathbf{p}_o$ of an object $o \in O$, and predicts keypoint locations $\mathbf{p}_o^a$ after applying each action $a \in \mathcal{A}$. Fig. 3 shows typical examples of $\mathbf{p}_{\texttt{chair}}$ and $\mathbf{p}_{\texttt{chair}}^a$ after applying `Push`, `Pull`, `RightPush` and `MoveAhead` actions. More specifically, the NIE predicts affine transformation matrices $m_o^a \in \mathbb{R}^{4\times4}$ corresponding to each object and each action. Then, we derive the $\mathbf{p}_o^a$ by translating and rotating the $\mathbf{p}_o$ via m_o^a in 3D space. Applying the affine transformation to the keypoints preserves the rigid body constraint while moving keypoints of the same object. The NIE summarizes both the extracted keypoints and the action-conditioned keypoints into an action-conditioned state feature r^a. In this way, the NIE provides possible outcomes resulting from each action to the policy network. Finally, given a goal representations g, the policy network utilizes both v and r^a to generate an action a for the agent.

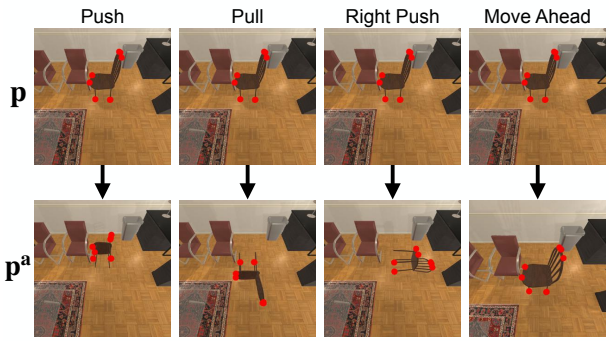

Figure 3: **Keypoint examples.** The top row shows object keypoints $\mathbf{p}_o$ and bottom row shows action-conditioned keypoints $\mathbf{p}_o^a$ resulted from `Push`, `Pull`, `RightPush` and `MoveAhead` actions. The keypoints are showon in red.

3.2. Neural Interaction Engine

The NIE operates by first extracting object keypoints $\mathbf{p} \in \mathbb{R}^{O\times(N\times3)}$, where N denotes the number of keypoints, O denotes the observed objects, and each $p \in \mathbb{R}^3$ describes a point in the three dimensional space, and then, based on these keypoints, predicting the action-conditioned keypoints $\mathbf{p}^a \in \mathbb{R}^{O\times|\mathcal{A}|\times(N\times3)}$ for each action a in the action space $\mathcal{A}$. The engine captures a summary of possible outcomes for each action and object. These summaries are used by the policy network to sample an action a.

As shown in Fig. 4, the input to NIE includes the observation i, which includes an RGB frame and a depth map, the visual representation v from the visual encoder, the object category embedding, and the action index embedding. The observation is first passed through a MaskRCNN [16] to obtain object segmentations. To extract the keypoints, we heuristically detect 8 corner points in an object segment as the keypoints for this object (see Sec. A for more details). We used a heuristic approach to find the keypoints, but any other keypoint detection approach (e.g., [20, 33]) could be used instead. Further, using the depth map and camera parameters of the agent, we back project the keypoints onto the 3-dimensional space.

To predict the outcome of each action, the NIE predicts affine transformation matrices for each object and action, as shown in the *Affine Transformation* module in Fig. 4. In practice, we first embed the keypoints $\mathbf{p}$ into hidden features and concatenate it with the object category embedding as well as the action index embedding. Then, we use an MLP to predict the affine transformation matrix $\mathbf{m} \in \mathbb{R}^{O\times|\mathcal{A}|\times4\times4}$ for all objects O and all actions in the action space $\mathcal{A}$. We translate and rotate the keypoints $\mathbf{p}$ according to $\mathbf{m}$ to obtain $\mathbf{p}^a$. Since each $m_o^a \in \mathbf{m}$ encodes the information associated with object category and the action

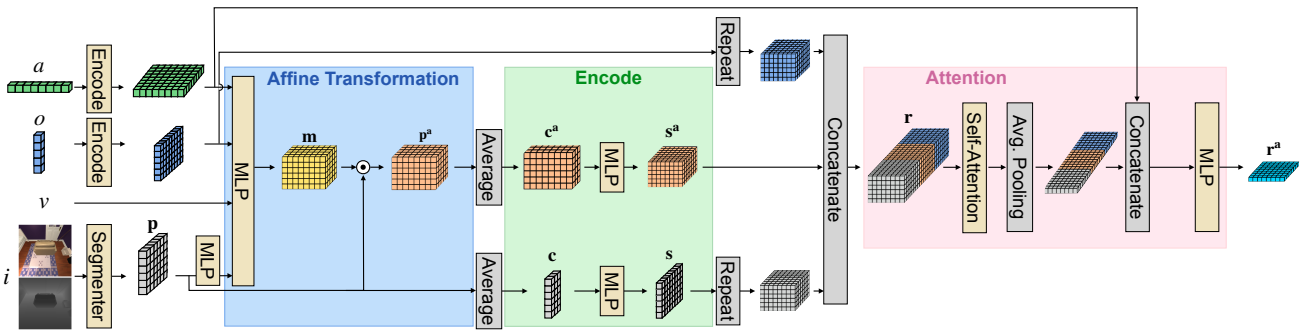

Figure 4: **Neural Interaction Engine.** The inputs to the neural interaction engine are action indices, object categories, visual representation v from the visual encoder, and visual observation i, which includes an RGB image and a depth map. After encoding each input modality, the engine uses an MLP to predict the affine transformation matrices to translate and rotate keypoints $\mathbf{p}$ to $\mathbf{p}^a$ corresponding to all objects and all actions. Then, the engine encodes the average of keypoints into hidden features $\mathbf{s}$ as well as $\mathbf{s}^a$. Finally, the engine utilizes a self-attention layer to summarize the hidden features into a semantic action-conditioned state representation $\mathbf{r}^a$.

a, the predicted keypoints not only contain semantic meaning, but also carry action-dependent information.

To encode keypoints and their corresponding action-conditioned keypoints, we first compute the center ($\mathbf{c}$ and $\mathbf{c}^a$) of both $\mathbf{p}$ and $\mathbf{p}^a$ by averaging the coordinates along each axis (i.e, $c_x = \frac{1}{N}\sum_{n=1}^{N} p_x^n$, $c_y = \frac{1}{N}\sum_{n=1}^{N} p_y^n$, $c_z = \frac{1}{N}\sum_{n=1}^{N} p_z^n$). Further, we employ a state encoder to encode $\mathbf{c}$ and $\mathbf{c}^a$ into hidden features ($\mathbf{s}$ and $\mathbf{s}^a$), as shown in the *Encode* module in Fig. 4.

The hidden features $\mathbf{s}$ and $\mathbf{s}^a$ are then concatenated with the object category embedding to construct a semantic action-conditioned state representation $\mathbf{r}$. Furthermore, we perform Self-Attention [34] on $\mathbf{r}$ over the object category axis and an Average-Pooling layer to obtain the action-conditioned state representation $\mathbf{r}^a$, as illustrated in the *Attention* module in Fig. 4. The reason for this step is not only to make the action-conditioned representation more compact, but also to directly associate it to each action.

Integrating NIE output into the Policy Network. We construct a global representation f by concatenating the goal representations g (e.g., target location encoding for the point navigation task), visual representation v, and action-dependent state features $\mathbf{r}^a$. The policy network takes f as the input and outputs a probability distribution over the action space. The agent samples an action from this distribution to execute in the environment.

3.3. Learning Objective

To train the model to learn the affine transformation matrix, we use the pose of an object before and after applying an action a in the environment to construct the ground truth affine transformation matrix. Then, we apply this ground truth affine transformation matrix to the keypoints $\mathbf{p}$ to obtain the ground truth action-conditioned keypoints $\mathbf{t}^a$. We

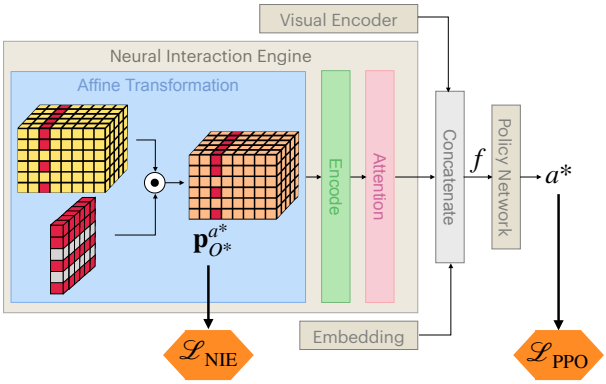

Figure 5: **Training pipeline.** The entire model is trained by $\mathcal{L}_{\text{PPO}}$ and the Affine Transformation module is trained by $\mathcal{L}_{\text{NIE}}$. However, the gradients backpropagated from $\mathcal{L}_{\text{NIE}}$ are only used to update the parameters corresponding to $\mathbf{p}^{a^*}_{O^*}$, where O^* are the observed object categories and a^* is the action taken by the agent. The tensors corresponding to $\mathbf{p}^{a^*}_{O^*}$ are highlighted in red.

cast the learning as a regression problem and use L1 loss to optimize NIE. The agent can only pick one action to execute at each timestamp. Hence it is not possible to obtain the ground truth action-conditioned keypoints $\mathbf{t}^a$ for all possible actions $a \in \mathcal{A}$. The agent only observes few objects among the object categories O, so we do not backpropagate the gradients back to the object categories that are not observed. As a result, during the training stage (as illustrated in Fig. 5), we only compute the loss for the executed action and backpropagate the gradients only through the path corresponding to a^*, the action that is actually executed by the agent and also the observed object categories $O^* \subset O$:

$$\mathcal{L}_{\text{NIE}} = L1(\mathbf{p}^{a^*}_{O^*}, \mathbf{t}^{a^*}_{O^*}). \quad (1)$$

Further, to learn the policy network, we employ the Proximal Policy Optimization (PPO) [30] to perform an on-policy reinforcement learning, as illustrated in Fig. 5. The overall learning objective is $\mathcal{L} = \mathcal{L}_{\text{PPO}} + \alpha\mathcal{L}_{\text{NIE}}$, where the $\alpha \geq 0$ is a hyperparameter controlling the relative importance of the NIE loss.

4. Experiments

To evaluate the effectiveness of the proposed Neural Interaction Engine, we evaluate it on two downstream tasks. In the following, we first describe the two downstream tasks. We then describe environment details and the datasets we have collected for training and evaluating the proposed framework. Further, we provide the implementation details in Sec. 4.1. In Sec. 4.2 and Sec. 4.3, we introduce our comparative baselines and variations of our model. Finally, we present quantitative and qualitative results in Sec. 4.4.

Downstream tasks. We consider two downstream tasks for our experiments:

- *ObsNav* – The goal of ObsNav is to move from a random starting location in a scene to specific coordinates while the path to the target point is blocked by obstacles on the floor. This is similar to PointNav [3] with the difference that the agent should move objects out of the way to reach the target.
- *ObjPlace* – The second downstream task that we consider is ObjPlace. The goal is to move an object on the floor from a random starting location to a specified coordinate in a scene. This task requires successive application of a force to an object while navigating towards the target point.

Successful completion of these tasks requires reasoning about the outcome of the agent actions while performing long-horizon planning. Therefore, they are suitable testbeds to evaluate our model.

Environment settings. In this work, we perform experiments on AI2-iTHOR [19] v2.7.2, which provides fairly accurate physical properties of objects. AI2-iTHOR is built using the Unity game engine which enables the simulation of physical agent-object and object-object interactions. In this environment, we consider actions $\mathcal{A} = \{$ `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, `LookDown`, `Push`, `Pull`, `RightPush`, `LeftPush`, `END`$\}$, where `MoveAhead` moves the agent ahead by 0.25 meters, `RotateRight` and `RotateLeft` change the agent's azimuth angle by ± 90 degrees, `LookUp` and `LookDown` rotate the agent's camera elevation angle by ± 30 degrees, the `Push`, `Pull`, `RightPush`, as well as `LeftPush` let the agent push (along $\pm z$ and $\pm x$ axis) the closest observed object by applying a force of 100 newtons. The agent issues the `END` to indicate that it has completed an episode. Fig. 3 shows four typical examples where the agent applies `Push`, `Pull`, `RightPush`, `LeftPush` actions. Finally, we set the height and width of RGB and depth images to 224. Thereby, the ground truth object segmentation used to learn the NIE is also of the same dimensions.

Data collection. We use *Kitchens*, *Living Rooms*, *Bedrooms*, and *Bathrooms* for our experiments (120 scenes in total). We follow the common practice for AI2-THOR wherein the first 20 scenes are used for training, the next 5 for validation, and the last 5 for testing in each scene category. To collect the datasets, we use 20 categories of objects such as `Chair`, `SideTable`, and `DogBed`. Please see Sec. B for the used objects. These objects are used as obstacles for *ObsNav* and as objects that should be displaced in *ObjPlace*. These objects are spawned on the floor for the downstream tasks. For each object category we have 5 different variations. We randomly select the first 4 variations to collect the training and validation data and use the 5th variation to collect the test data.

To generate the dataset for *ObsNav*, we utilize an undirected graph to compute the path from the agent's starting location to the target location. Then, we randomly spawn an object to block the path. To ensure that there is no way that the agent can directly reach the target location without moving an object, we repeat this process until there is no path between the agent's starting location (source node) and target location (end node). The top row in Fig. 6 shows five examples in this dataset.

To generate the dataset for *ObjPlace*, we first create a yellow mark at a random location on the floor in a scene to indicate the target location. We then spawn an object at another random location, which is at least 2 meters away from the target location. In total, we collect 10k training instances, 2.5k validation instances, and 2.5k testing instances for each task. The bottom row in Fig. 6 shows five examples in this dataset.

4.1. Implementation details

In this work, we use the AllenAct [37] framework to conduct experiments. We train our model using both $\mathcal{L}_{\text{PPO}}$ and $\mathcal{L}_{\text{NIE}}$ simultaneously. We set the α parameter in Section 3.3 to 3. We discuss the effect of α on the performance in Sec. 4.4. For *ObsNav*/*ObjPlace*, an episode is successful if the agent invokes `END` while the agent/object reaches a position within 0.2 meters of the target position. During the training stage, we perform the on-policy reinforcement learning (PPO) with 80 processes simultaneously. We use Adam with initial learning rate of $3 \cdot 10^{-4}$ which decays linearly to 0 during training. We set the standard RL reward discounting parameter γ to 0.99, λ_{gae} to 0.95, and number of update steps to 30 for $\mathcal{L}_{\text{PPO}}$. The gradients Δ are clipped to satisfy $|\Delta| <= 0.5$. We train the policy for 10 million steps and evaluate the model every 1 million steps.

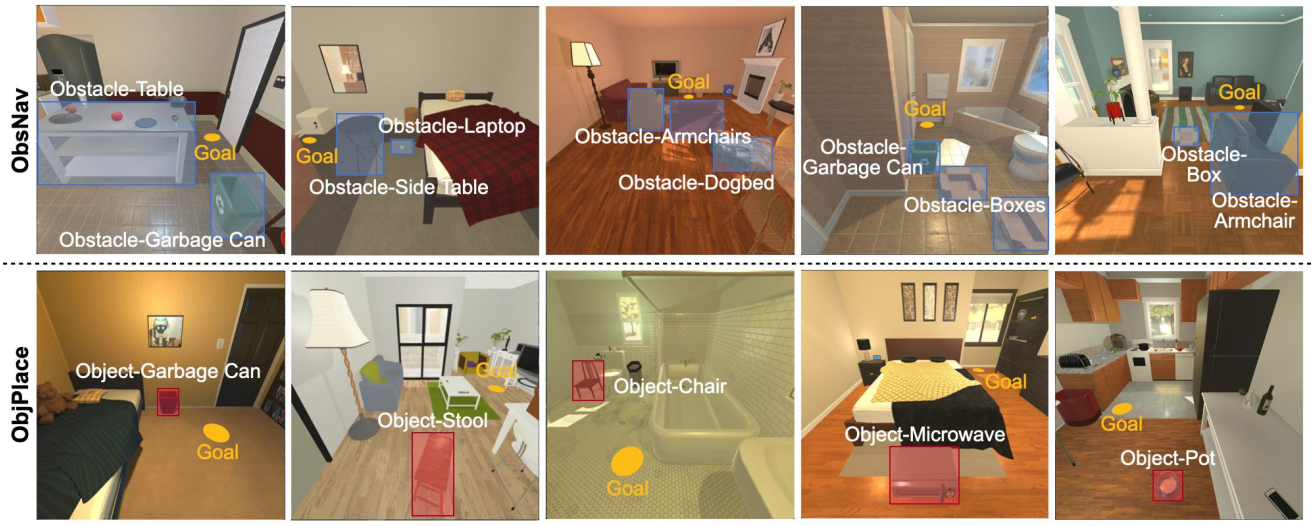

Figure 6: **Dataset examples.** Top: five examples in *ObsNav* dataset, where the blue boxes are obstacles and the yellow circle is the target position. Bottom: five examples in *ObjPlace* dataset, where the red boxes are the object that should be displaced and the yellow circle is the target place.

During the training stage, we use the ground truth object mask provided by the environment, while in the testing stage, we employ a pre-trained MaskRCNN [16] to extract the segmentation. The number of output classes for both ground truth segmentation and MaskRCNN is 21, including 20 used objects and a background class. We use [42] to pre-train the MaskRCNN (ResNet-50 with FPN) on our training scenes with 8k images for 10 epochs. Please see Sec. C for more details about the qualitative results generated by the MaskRCNN on our validation scenes.

Model architecture. Because the visual observation i includes an RGB image and a depth image, we employ two different CNNs, with different input number of channels, in the Visual Encoder to handle these two observations separately. After the CNNs extract features from both observations, we use a linear layer to fuse the two features together. In both tasks, we provide the observation from a GPS sensor to the policy network. The GPS's observation is a coordinate of the target position for *ObsNav* or the target place for *ObjPlace*. In addition to the GPS's observation, we employ a look-up embedding to encode the category of the target object for *ObjPlace*. The Encode and MLP shown in Fig. 4 are a look-up embedding layer and a multi-layer perceptron, respectively. The Self-Attention layer has three MLPs as well to handle the key, query, and value embedding. Our policy network consists of a GRU state encoder, a linear layer for the actor (policy), and a linear layer for the critic (value). Please refer to Sec. D for more details about each model components such as the number of layers and hidden dimension.

Reward shaping. We consider a task successful if the agent invokes the `END` when the agent achieves the goal. For *ObsNav*, the goal is to reach within a certain distance (0.2 meters) to the target location and for *ObjPlace*, the object should have overlap with the yellow target mark. If the agent succeeds in an episode, we provide a reward of $+10$. We find reward shaping [27] important to learn the policy in the two studied tasks. We implement reward shaping for each task as follows:

• *ObsNav*: Similar to [29], we implement the reward shaping based on geodesic distance. We provide a reward to the agent after it takes an action based on the change in the geodesic distance between the current agent position and the target position. If the agent takes an action resulting in a decrease of the geodesic distance, the agent receives the decreased amount as the reward. Otherwise, if the taken action causing an increase in the geodesic distance, the agent receives the amount of increase as a penalty. There are obstacles blocking the paths to the destination, so we also encourage the agent to take actions to move the obstacles out of the way. Therefore, the environment provides a -0.5 penalty if the agent takes any action that blocks a path between the agent and the goal and conversely, a 0.5 reward if the agent's action opens a new path to the goal. We let $r_{\text{dis/appear}} = -0.5$ if the agent's action resulted in blocking a path, $r_{\text{dis/appear}} = 0.5$ if the agent's action opened a path, and $r_{\text{dis/appear}} = 0$ otherwise.

• *ObjPlace*: For this task, we perform reward shaping only based on the geodesic distance. We provide a reward to the agent after it takes an action according to the change in the geodesic distance between the current object position and the target position.

To encourage the agent to finish the task as quickly as possible, we also add a small penalty -0.01 at each step.

As a result, the total reward at step t is:

$$r_t = \begin{cases} r_s + r_{\text{dis/appear}} + d_{t-1} - d_t + p & \text{if goal is reached,} \\ r_{\text{dis/appear}} + d_{t-1} - d_t + p & \text{otherwise,} \end{cases}$$

where r_s is set to 10, d_t denotes the geodesic distance between the agent (object) and the target position at step t, and p is the step penalty which equals -0.01 and -0.002 for *ObsNav* and *ObjPlace*, respectively. Note that the $r_{\text{dis/appear}}$ is removed in the *ObjPlace* task. However, adding $r_{\text{dis/appear}}$ is essential to the *ObsNav* task, since the policy network with visual observation i but without $r_{\text{dis/appear}}$ does not generalize to the unseen environments even after 10 million steps of training.

4.2. Baselines

We compare our model with the following baseline methods. Each baseline uses the same visual encoder and policy network unless stated otherwise.

PPO. This baseline is a Reinforcement Learning based approach that has a visual encoder to extract features from visual observation i and an embedding layer to encode the GPS readings. The model is trained by Proximal Policy Optimization [30] and we use the same learning hyparameters mentioned in Sec. 4.1 to train using this method.

RGB-D and object segmentation input (RGB-D-S). This baseline includes a segmentation image as well as the visual observation i. We extend the Visual Encoder by another CNN to extract features from the segmentation image. As mentioned in Sec. 4.1, we use the ground truth segmentation during the training stage and the results generated by MaskRCNN (fine-tuned on our data) during the evaluation.

RGB-D and keypoints input (RGB-D-K). To understand if the keypoints representation extracted from object segmentation is more meaningful than a pure segmentation image input, we implement this baseline by including the keypoints extracted by the same heuristic corner detector (Sec. 3.2) used in our model as an additional input. To encode the keypoints, we use the same model architecture as the NIE module to obtain the semantic action-conditioned state representation $\mathbf{r}^a$ as well. However, the $\mathcal{L}_{\text{NIE}}$ is not used to learn the NIE module in this baseline. The parameters are updated by the gradients from $\mathcal{L}_{\text{PPO}}$ only.

PPO + auxiliary loss. We implement a baseline based on CPC|A [15] to facilitate the policy learning upon the PPO baseline. During the training stage, the CPC|A utilizes Contrastive Predictive Coding as an auxiliary loss to perform predictive representation learning. To have a fair comparison, we use a GRU with the same hidden size and only predict one time step in the future.

4.3. Ablations

To perform ablation studies, we evaluate the following variations of our NIE model.

Methods	**SR (%)** ↑	**FDT** (m) ↓	**SPL** ↑
Baselines:			
PPO [30]	67.1	0.605	25.7
RGB-D-S	62.8	0.499	25.0
RGB-D-K	70.9	0.459	25.8
CPC|A [15]	73.8	0.370	29.8
NIE **(ours)**	**80.0**	0.304	**31.3**
Ablations:			
NIE w/o VO	72.7	0.375	29.2
NIE w/ $1 \times \mathcal{L}_{\text{NIE}}$	74.1	0.377	29.7
NIE w/ $10 \times \mathcal{L}_{\text{NIE}}$	78.2	**0.278**	31.0

Table 1: **ObsNav results.** We show the result of our method (referred to as 'NIE') along with baselines and ablations of our model. We use ↑ and ↓ to denote if larger or smaller values are preferred. We repeat the experiments three times and report the average.

NIE w/o visual observations. To understand if the visual observation i can help the prediction of affine transformation matrices and the action-conditioned keypoints $\mathbf{p}^a$, we implement this model by removing the visual input from the NIE. Therefore, the NIE only takes the keypoints in coordinate representation with action indices as well as object categories. We use the same hyperparameters and optimization approach mentioned in Sec. 4.1 to train this model.

NIE w/ $1 \times \mathcal{L}_{\text{NIE}}$**.** We decrease α, which is used to balance the $\mathcal{L}_{\text{NIE}}$ and $\mathcal{L}_{\text{PPO}}$. This provides us with an insight about the importance of $\mathcal{L}_{\text{NIE}}$ to learn the entire model.

NIE w/ $10 \times \mathcal{L}_{\text{NIE}}$**.** In this ablation study, we increase the α, which is used to balance the $\mathcal{L}_{\text{NIE}}$ and $\mathcal{L}_{\text{PPO}}$, to 10. This study shows if a large value of α would have a negative impact on the final performance.

4.4. Results

Evaluation Metrics. We evaluate all models by *Success Rate (SR)*, *Final Distance to Target (FDT)*, and *Success weighted by Path Length (SPL)* [2] for both tasks. *SR* is the ratio of the number of successful episodes to the total number of episodes, *FDT* is the average distance between agent/object and the target position as the agent issues `END` or an episode reaches the maximum number of allowed steps (500), and the *SPL* is defined as $\frac{1}{N}\sum_{n=1}^{N} S_n \frac{L_n}{max(P_n, L_n)}$, where N is the number of episodes, S_n denotes a binary indicator of success in the episode n, P_n is the path length, and L_n is the shortest path distance in episode n.

ObsNav. The quantitative results of the *ObsNav* task are shown in Table 1. Our method outperforms the baselines in all three metrics, which justifies the effect of using the NIE model. The performance drops for 'NIE w/o VO' ablations, which shows that visual information is required to estimate the location of objects. For example, if an object is pushed against a wall, the visual information helps to reason that

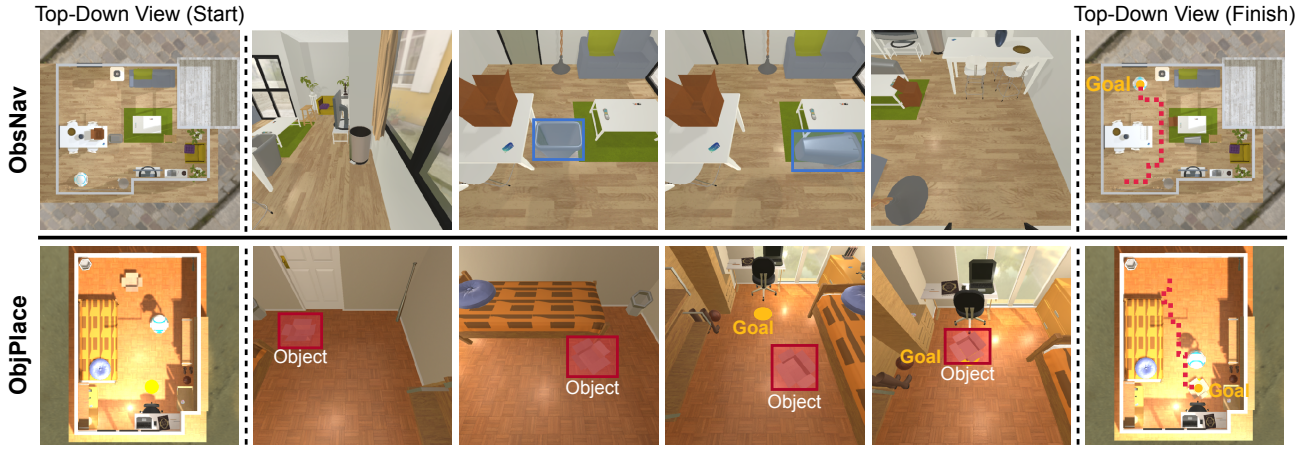

Figure 7: **Qualitative results.** Top: An example of the *ObsNav* task is shown. The blue box is the obstacle the agent should move away to unblock the path (the blue marking is just for visualization purposes and not visible to the agent). The agent's movement is shown by a dashed trajectory in red in the rightmost image. Bottom: An example of the *ObjPlace* task, where the red box is the object that should be displaced and the orange circle is the target location. The object's movement is shown by a trajectory in red color.

Methods	**SR (%)** ↑	**FDT** (m) ↓	**SPL** ↑
Baselines:			
PPO [30]	1.2	3.18	0.85
RGB-D-S	1.2	3.15	0.85
RGB-D-K	1.3	2.84	0.88
CPC|A [15]	12.0	2.35	9.3
NIE **(ours)**	**17.5**	2.22	**14.2**
Ablations:			
NIE w/o VO	0.8	3.07	0.41
NIE w/ $1 \times \mathcal{L}_{\text{NIE}}$	15.3	**2.11**	13.1
NIE w/ $10 \times \mathcal{L}_{\text{NIE}}$	13.6	2.26	11.5

Table 2: **ObjPlace results.** We show the result of our method (referred to as 'NIE') along with baselines and ablations of our model. We use ↑ and ↓ to denote if larger or smaller values are preferred. We repeat the experiments three times and report the average.

the object will not move. It is not feasible to make such predictions just by using the keypoint information alone. Our results on 'NIE w/ $1 \times \mathcal{L}_{\text{NIE}}$' and 'NIE w/ $10 \times \mathcal{L}_{\text{NIE}}$' show that completely relying on the NIE model is not sufficient and we need exploration as well. On the other hand, exploration alone is not sufficient. Therefore, a good balance between future prediction and exploration is required.

ObjPlace. The results are shown in Table 2. As shown, there is a huge difference between the baseline models and our model. We investigated the reason for this huge gap. Most of the time the baseline agent pushes other objects as well and eventually blocks the path towards the target.

Qualitative Results. We show qualitative results in Fig. 7. The top row shows a successful episode of ObsNav, where the agent pushes the garbage can away to unblock the path. The bottom row show an example of the ObjPlace task, where the agent moves the box toward the goal position. It is interesting to note that the agent goes around the object of interest so it can push it towards the target location. We provide a supplementary video to show more successful and failure cases. We also provide qualitative results of keypoint prediction in Sec. E. We show how well the NIE model predicts the future location of keypoints conditioned on the actions.

5. Conclusion

We study the problem of predicting the outcome of actions in the context of embodied visual navigation tasks. We propose Neural Interaction Engine (NIE) to encode the changes to the environment caused by navigation and interaction actions of the agents. We incorporate NIE into a policy network and show its effectiveness in two downstream tasks that require long-horizon planning. The goal of the first task is to reach a target point in an environment while the paths to the target are blocked. The second task requires navigating to a target point while pushing an object. Our evaluations show the effectiveness of the NIE model in both scenarios, where we achieve significant improvements over the methods without the capability of predicting the effect of actions on the surrounding environment.

Acknowledgements.

We thank members from RAIVN Lab at the University of Washington and PRIOR team at Allen Institute for AI for valuable feedbacks on early versions of this project. This work is in part supported by NSF IIS 1652052, IIS 17303166, DARPA N66001-19-2-4031, DARPA W911NF-15-1-0543 and gifts from Allen Institute for AI.

References

[1] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *NeurIPS*, 2016. 2

[2] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. In *arXiv*, 2018. 7

[3] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *arXiv*, 2018. 1, 2, 5

[4] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *ArXiv*, 2020. 2

[5] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, A. Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv*, 2020. 1, 2

[6] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *ICRA*, 2017. 2

[7] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *ICLR*, 2017. 2

[8] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *NeurIPS*, 2020. 1, 2

[9] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *ICLR*, 2020. 2

[10] Heming Du, Xin Yu, and Liang Zheng. Learning object relation graph and tentative policy for visual navigation. In *ECCV*, 2020. 2

[11] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv*, 2018. 2

[12] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *CoRL*, 2017. 2

[13] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *ICRA*, 2017. 2

[14] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning predictive visual models of physics for playing billiards. In *ICLR*, 2016. 2

[15] Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A Pires, and Rémi Munos. Neural predictive belief representations. In *arXiv preprint arXiv:1811.06407*, 2018. 7, 8

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 3, 6

[17] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. In *ICLR*, 2019. 2

[18] Peter Karkus, Mehdi Mirza, Arthur Guez, Andrew Jaegle, Timothy Lillicrap, Lars Buesing, Nicolas Heess, and Theophane Weber. Beyond tabula-rasa: a modular reinforcement learning approach for physically embedded 3d sokoban. *ArXiv*, 2020. 2

[19] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. In *Arxiv*, 2017. 2, 5

[20] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. In *NeurIPS*, 2019. 3

[21] Juekun Li, Wee Sun Lee, and David Hsu. Push-net: Deep planar pushing for objects with unknown physical properties. In *RSS*, 2018. 2

[22] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel L.K. Yamins, Jiajun Wu, Joshua B. Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models. In *ICLR*, 2020. 2

[23] Mehdi Mirza, Andrew Jaegle, Jonathan J Hunt, Arthur Guez, Saran Tunyasuvunakool, Alistair Muldal, Théophane Weber, Peter Karkus, Sébastien Racanière, Lars Buesing, et al. Physically embedded planning problems: New challenges for reinforcement learning. *ArXiv*, 2020. 2

[24] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian image understanding: Unfolding the dynamics of objects in static images. In *CVPR*, 2016. 2

[25] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "what happens if..." learning to predict the effect of forces in images. In *ECCV*, 2016. 2

[26] Iman Nematollahi, Oier Mees, Lukas Hermann, and Wolfram Burgard. Hindsight for foresight: Unsupervised structured dynamics models from physical interaction. In *IROS*, 2020. 2

[27] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999. 6

[28] Fabian Paus, Teng Huang, and Tamim Asfour. Predicting pushing action effects on spatial object relations by learning internal prediction models. In *ICRA*, 2020. 2

[29] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *ICCV*, 2019. 1, 2, 6

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. 5, 7, 8

[31] Michael Stilman and James Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Humanoids*, 2004. 2

[32] Michael Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. *IJRR*, 2008. 2

[33] Supasorn Suwajanakorn, Noah Snavely, Jonathan J Tompson, and Mohammad Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *NeurIPS*, 2018. 3

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 4

[35] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. Sfm-net: Learning of structure and motion from video. *arXiv*, 2017. 2

[36] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *NeurIPS*, 2017. 2

[37] Luca Weihs, Jordi Salvador, Klemen Kotar, Unnat Jain, Kuo-Hao Zeng, Roozbeh Mottaghi, and Aniruddha Kembhavi. Allenact: A framework for embodied ai research. In *arXiv*, 2020. 5

[38] 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

[39] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In *CVPR*, 2019. 1, 2

[40] Jiajun Wu, Joseph J Lim, Hongyi Zhang, Joshua B Tenenbaum, and William T Freeman. Physics 101: Learning physical object properties from unlabeled videos. In *BMVC*, 2016. 2

[41] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual deanimation. In *NeurIPS*, 2017. 2

[42] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 6

[43] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. In *ICRA*, 2021. 2

[44] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. In *ICRA*, 2020. 2, 3

[45] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B. Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. In *RSS*, 2019. 2

[46] Kuo-Hao Zeng, Roozbeh Mottaghi, Luca Weihs, and Ali Farhadi. Visual reaction: Learning to play catch with your drone. In *CVPR*, 2020. 2

[47] Kuo-Hao Zeng, William B Shen, De-An Huang, Min Sun, and Juan Carlos Niebles. Visual forecasting by imitating dynamics in natural sequences. In *ICCV*, 2017. 2

[48] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 1

A. Heuristic corner detector

Fig. 9 (a) shows our heuristic keypoints detector pipeline. More specifically, we first use an object segmentation model to obtain the segmentation M corresponding to object $o = \texttt{GarbageCan}$. Then, we apply a heuristic corner detector to detect 8 corner points. Note that we use the ground truth segmentation of each object in the training stage, while in the testing stage, we utilize a pretrained MaskRCNN (Sec. C) to obtain the object segmentation. Further we present the details of our heuristic corner detector in Fig. 9 (b), where the 8 corner points are obtained by 8 different criteria and each of the points has to be inside the segmentation M:

$$
\begin{aligned}
p_1 &= \max_{x,\ p_{x,y} \in M} p_{x,y} \\
p_2 &= \max_{y,\ p_{x,y} \in M} p_{x,y} \\
p_3 &= \min_{x,\ p_{x,y} \in M} p_{x,y} \\
p_4 &= \min_{y,\ p_{x,y} \in M} p_{x,y} \\
p_5 &= \max_{x+y,\ p_{x,y} \in M} p_{x,y} \\
p_6 &= \min_{x+y,\ p_{x,y} \in M} p_{x,y} \\
p_7 &= \max_{x-y,\ p_{x,y} \in M} p_{x,y} \\
p_8 &= \min_{x-y,\ p_{x,y} \in M} p_{x,y},
\end{aligned}
$$

where (x, y) is an image coordinate, and M denotes the object segmentation. Based on this heuristic corner detector, we are able to get reliable keypoints from object segmentation. More keypoint examples obtained by our heuristic keypoints detector are shown in Fig. 8.

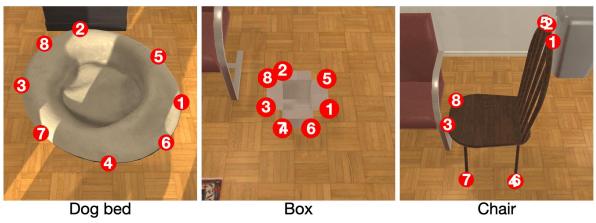

Figure 8: **Keypoints examples.** Examples keypoints obtained by our keypoint detector.

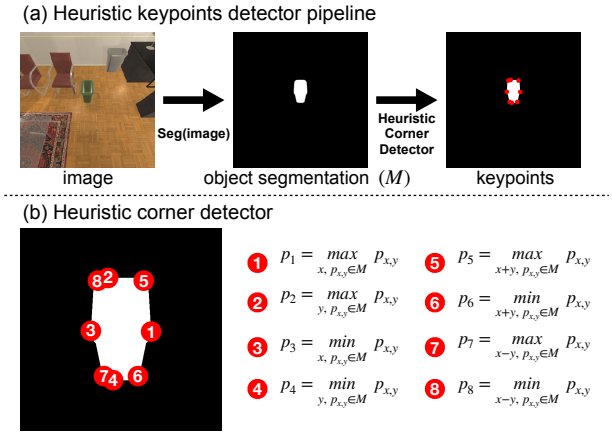

Figure 9: **Keypoint detector details.** (a) Heuristic keypoint detector pipeline. (b) Heurisitc corner detector.

B. Complete list of objects

We use 20 objects for the experiments: *alarm clock*, *apple*, *armchair*, *box*, *bread*, *chair*, *desk*, *dining table*, *dog bed*, *garbage can*, *laptop*, *lettuce*, *microwave*, *pillow*, *pot*, *side table*, *sofa*, *stool*, *television* and *tomato*.

C. MaskRCNN results

We evaluate our pretrained MaskRCNN (ResNet-50 with FPN) on our testing scenes with $\approx$ 2k images. The checkpoint at the 10th epoch achieves 47.4AP and 64.3AP$_{50}$. Fig. 12 shows qualitative results on 20 used objects in one of the testing scenes `LivingRoom227`.

D. Details of the model architecture

Fig.11 and Fig. 10 summarize the details of the architecture for visual encoder, goal embedding, policy network, and NIE model.

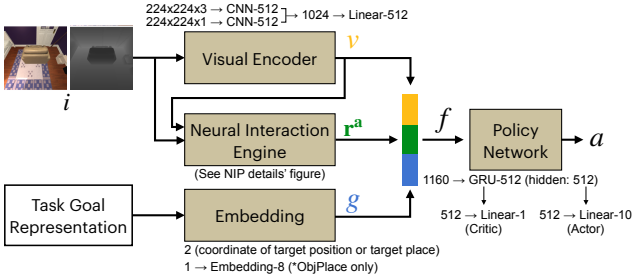

Figure 10: **Detailed architecture of the visual encoder, goal embedding, and policy network.**

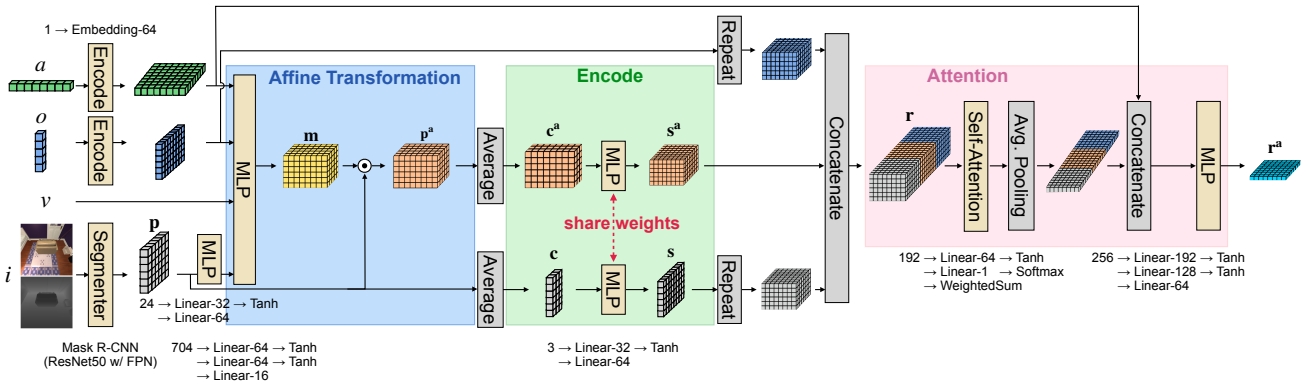

Figure 11: **Detailed architecture of the NIE model.**

E. Action-conditioned keypoints $\mathbf{p}^a$ results

We evaluate our action-conditioned keypoints $\mathbf{p}^a$ prediction on the testing set. Our model achieves 0.148 and 0.114 L1 loss estimation over 8 keypoints on the *ObsNav* and *ObjPlace*, respectively. We found the model performs worse in the *ObsNav* because there are more objects (e.g., obstacles) in this task. Fig. 13 shows the qualitative results of our action-conditioned keypoint prediction.

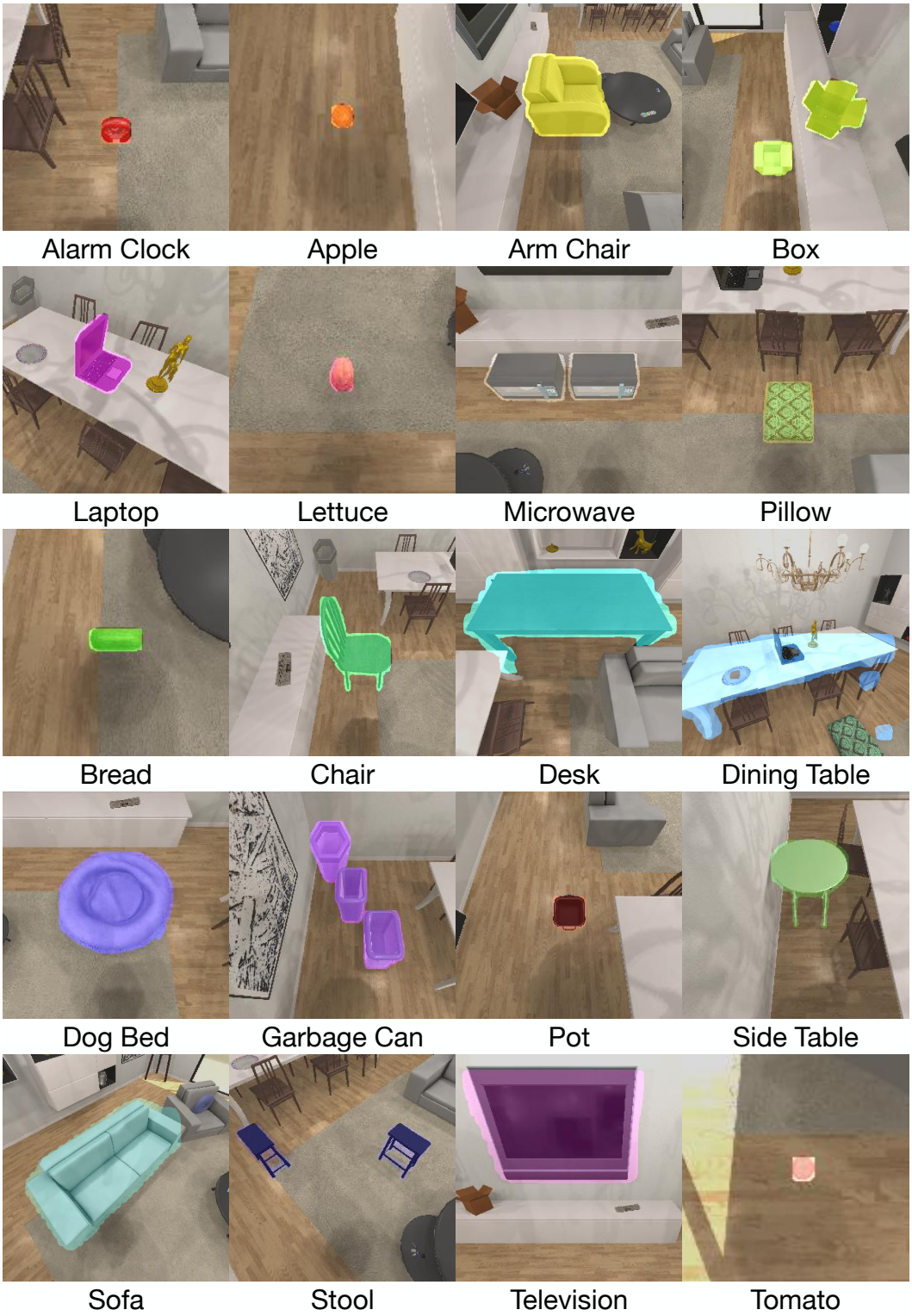

Figure 12: **MaskRCNN's qualitative results on** 20 **used objects.** We randomly spawn 20 objects in the testing scene `LivingRoom227` and apply the pretrained MaskRCNN to obtain the segmentation results. The object prediction score is set to 0.5 and the segmentation probability is set to 0.1.

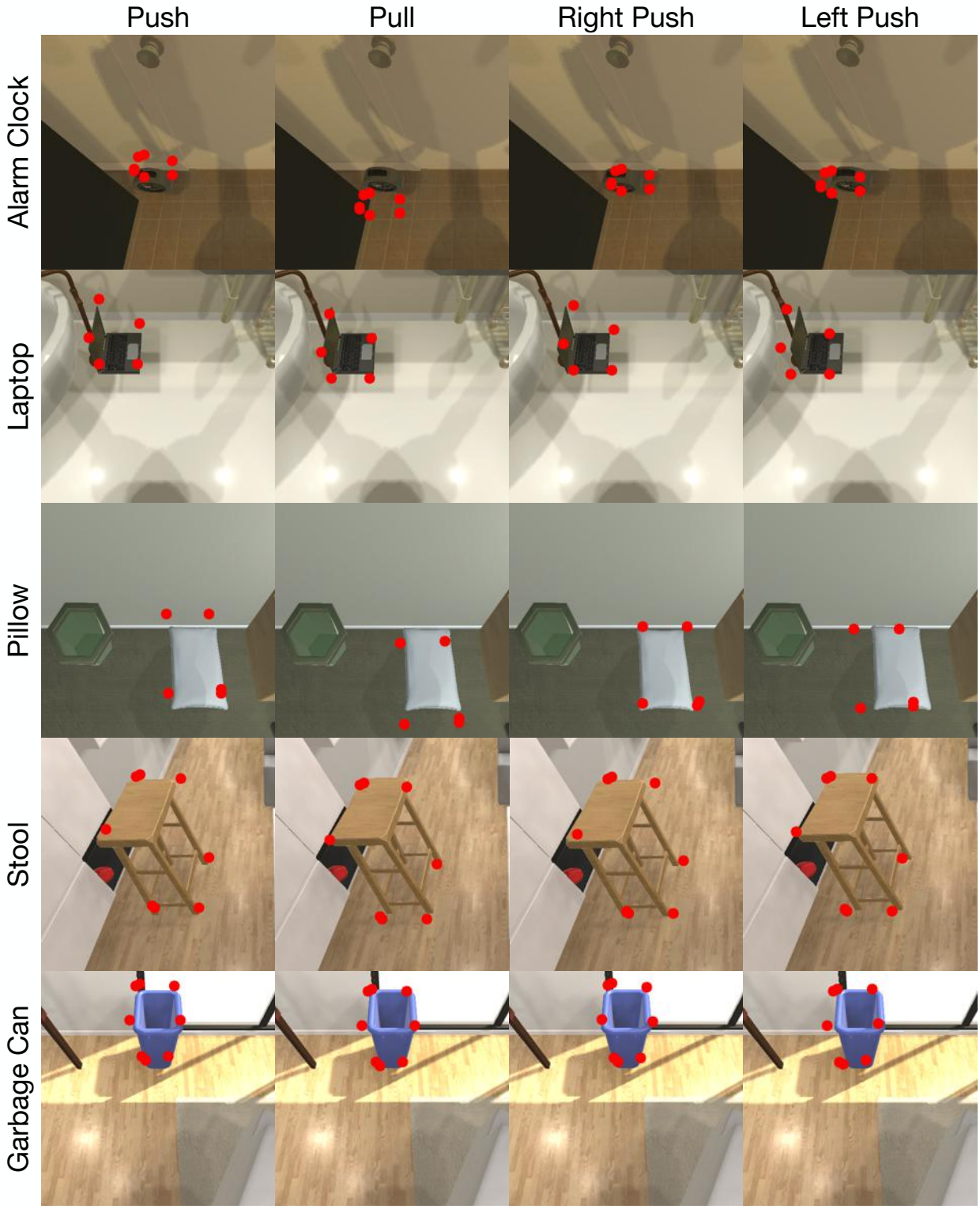

Figure 13: **Qualitative results of action-conditioned keypoints $\mathbf{p}^a$ prediction.** We show our action-conditioned keypoints $\mathbf{p}^a$ prediction results over 4 actions on 5 objects in 4 different testing scene (from top to bottom: `Kitchen27`, `Bathroom430`, `Bedroom328`, and `LivingRoom227`). The predicted keypoints are shown in red color.