Visual Forecasting for Interactive Embodied Agent

Kuo-Hao Zeng

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Ali Farhadi, Chair

Roozbeh Mottaghi

Luca Weihs

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Visual Forecasting for Interactive Embodied Agent

Kuo-Hao Zeng

Chair of the Supervisory Committee:
Professor Ali Farhadi
Computer Science and Engineering

A hallmark of human intelligence is the ability to plan by predicting the future. Equipping artificial agents with such capability is essential for many fields, especially when the agents have to interact with a dynamic, uncertain environment. This thesis explores how to integrate visual forecasting models into embodied agents. Firstly, we introduce how to perform efficient planning based on trajectory forecasting. Specifically, we developed a drone agent to play catch in a simulated environment. The policy realizes efficient planning by using a model-predictive controller (MPC) with a learnable action sampler. The goal is to forecast the trajectory of the object of interest and plan accordingly. Secondly, we present how to achieve an effective interactive visual navigation, in which the agent has to accomplish tasks by changing the environment. We propose a Neural Interaction Engine (NIE) to enable action-centric object state anticipation. The agent decides based on the predicted one-step forward and action-dependent object state with the NIE, allowing it to solve tasks more effectively. Finally, we utilize visual forecasting to adapt agents to unexpected action drifts. To this end, we introduce Action Adaptive Policy (AAP) to enable agents to adapt to unseen drifts at inference. The key idea is to learn and leverage an action-impact embedding using visual forecasting formulation. In this way, the agent learns how to encode the action-impact embedding on-the-fly to adapt to unseen drifts. The experimental results show that our approach consistently performs better across unseen drifts and even works well when some actions are disabled at inference.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I sincerely thank my academic advisor, Professor Ali Farhadi. Ali continues to inspire me with exciting and brilliant research ideas. He often provides me with a high-level overview of the entire project. We set up a roadmap and milestones during our meetings while leaving enough flexibility to adapt quickly and avoid getting stuck at a certain point. Together, we examine the data, experimental settings, and results, discussing the possible reasons for the results and the next steps I should take. Our discussions are always constructive and invaluable.

I am also fortunate to have been co-advised by Professor Roozbeh Mottaghi. Roozbeh frequently discusses experimental settings and model design details with me or shares the latest information regarding our research direction. In addition, he has provided suggestions from his standpoint regarding my career plan and shared his experiences as we discussed specific cases. I appreciate all his guidance and support.

I would like to extend my gratitude to Dr. Luca Weihs, who always offers his ideas, opinions, and suggestions regarding my projects, particularly when I encounter specific challenges or get stuck on certain issues. His suggestions are always detailed and helpful, and this thesis could not exist without his invaluable guidance and support.

I would also like to thank my committee members, Professor Dieter Fox and Professor Maryam Fazel, for their insightful discussions during the defense and thoughtful comments and suggestions.

Next, I want to thank all my labmates and co-authors who provided their expertise and contributions throughout various projects. Their talent and passion kept me working harder and smarter.

I acknowledge the financial support that made this research possible, including the Paul G. Allen School First-Year Ph.D. Fellowship from the CSE at UW and the J.P. Morgan

2021 Ph.D. Fellowship. Additionally, I would like to thank the Allen Institute for AI for their computing and resource support.

I want to thank my parents, cats, wife, and son for their endless support and confidence in me. In particular, I would like to thank my wife, Shih-Han Chou. Without her support, I would not have conquered various obstacles and completed this thesis.

Finally, I express my deepest gratitude to Elise deGoede Dorough, the director of graduate student services at CSE, for her professional support for students. Without her help with my mental stress, registration issues, and funding problems, I could never have accomplished the many milestones during my CSE studies.

DEDICATION

to my dear wife, Shih-Han, my dear son, Logan, and my parents

Chapter 1

INTRODUCTION

Learning an embodied agent to interact with an environment only with visual observations is challenging, especially for promptly accomplishing a task in an interactive scene with unexpected action outcomes. Learning a dynamics model of the agent and the environment via visual observations could be the first step to tackling the challenge. With the learned dynamics model, the agent can roll out possible future states for planning. Then, with enough future rollouts, the agent gains reasonable confidence to prepare for incoming situations. Moreover, the agent can even learn a policy to solve the a defined task *in* the learned dynamics model [77, 79, 81, 82]. Thus, learning a high-fidelity dynamics model for Embodied AI has been an active research regime [47, 64, 119, 152, 252] for decades.

On the other side, learning a model-free policy to drive an embodied agent from visual observations has presented impressive results [8, 42, 45, 54] recently. Although many works [78, 128, 139, 178, 179] show that the model-free policy usually performs better than the model-based policy, it is notorious for the generalization challenge [50, 98, 165, 213, 238, 248], the sample inefficiency issue [10, 41, 60, 71, 118, 147], and the robustness problem [10, 50, 141, 163, 170]. Moreover, a single target task often needs high-quality reward signals to learn a satisfying policy. From this perspective, this thesis devotes to marrying the planning through the visual dynamics model approach with the model-free policy learning framework to overcome the mentioned challenges. Specifically, we aim to bring advantages from each other to develop our policy network and learning objectives. As shown in Fig. 1.1, we aim to learn a policy that can forecast future states and decide whose impact will allow the agent to achieve its goal more *efficiently*, *effectively*, and *robustly*. In this thesis, our key idea is to forecast the future via the visual dynamics model, then incorporate the information about future rollouts to modulate the planning process in the model-free policy network.

Many past works [51, 56, 79, 81, 82, 91, 115, 147, 165] share the same goal with us.

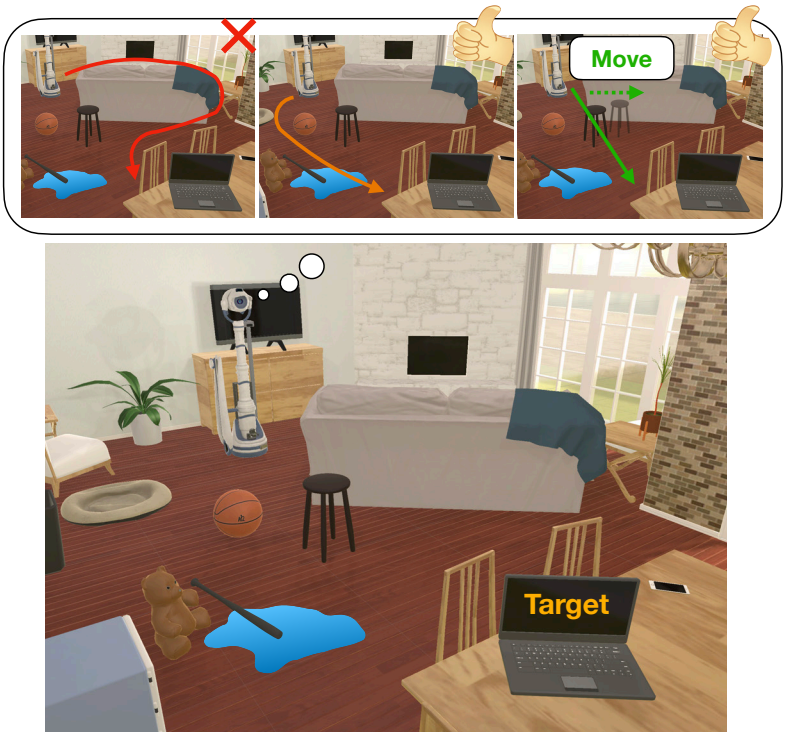

Figure 1.1: **Visual Forecasting for Interactive Embodied Agent.** Our key idea is to first perform visual forecasting via the dynamics model and then modulate the planning process in the model-free policy network via future information. Our framework has several characteristics: (1) *Efficiency*, the agent can select the most efficient route (*e.g.*, orange color) to the target, (2) *Effectiveness*, the agent avoids the route passing the other side of the sofa because it can not move the clustered objects away, nor bypass them (*e.g.*, red color). Instead, the agent recognizes it can push the chair to the free space behind the sofa, so it creates a new route in this way to solve the task (*e.g.*, green color), and (3) *Robustness*, the agent can adapt the policy to the unexpected action outcomes according to the observed state-changes (*e.g.*, orange color).

However, they generally (1) develop the agent in relatively simple and unrealistic environments [77, 79, 81, 115, 147, 165] or (2) concentrate on either learning a dynamics model [51, 56, 131] only or a policy network [129, 207] solely and further apply a hand-crafted

policy network/dynamics model to evaluate the learned dynamics model/policy network. In this context, it remains unclear how to learn the dynamics model and then perform planning *simultaneously* and *accordingly* to solve an embodiment task in a realistic environment. Therefore, this thesis focuses on:

- Learning a visual dynamics model to foresee the future states to enable efficient planning;
- Equipping the model-free policy with the forecasting results for effective decision-making;
- Learning an adaptable policy network with an action-dependent future prediction to robustly solve a task with unexpected situations at inference.

As shown in Fig. 1.1, when the task for the agent is to pick up the computer on the table, our agent can make multiple plans based on the agent's and the environment's future states. It then selects a plan to accomplish the task. Note that our future predictions are associated with available actions in the action space. For instance, if the agent can move the chair, there should be an option where the agent moves the chair out of the way to find a direct route to avoid the water on the ground. Therefore, our framework has several characteristics: (1) *Efficiency*, the agent can select the most efficient route (*e.g.*, orange color) to the target. Although the agent has to move on the water (*i.e.*, environmental conditions change), it prevents manipulating the chair to save energy; (2) *Effectiveness*, the agent avoids the route passing the other side of the sofa because it can not move the clustered objects away, nor bypass them (*e.g.*, red color). Instead, the agent recognizes it can push the chair to the free space behind the sofa, so it creates a new route in this way to solve the task (*e.g.*, green color); (3) *Robustness*, if the agent has to navigate through the water on the ground, it can adapt the policy to the unexpected action outcomes according to the observed state-changes (*e.g.*, orange color)[1]. In contrast to prior works, we evaluate our framework in physics-enabled and visually rich AI2-THOR [106] and Habitat [197] environments.

To outfit the embodied agents with visual forecasting capability, we need to understand (1) the current advancement of visual forecasting models, (2) how to benefit a policy with

[1]This state-changes formulation is based on the visual forecasting formulation. Please find Chap. 4 for more details.

forecasting results, and (3) how to tailor an embodied agent for some interested tasks. In the following sections, we first discuss several research directions *only* focusing on forecasting the future by *passive* visual input and point out why it is essential to have visual forecasting for an embodied agent (*i.e.*, active data) in Sec. 1.1. Further, we present the primary difference between traditional planning frameworks with forecasting and our approach in Sec. 1.2. Then, we introduce our three studied problems to showcase how to develop a more efficient, effective, and robust policy with visual forecasting in Sec. 1.3. Finally, we outline the thesis overview in Sec. 1.4.

1.1 Visual Forecasting for Policy Learning

Over the years, various methods have been proposed for the problem of visual forecasting [6, 55, 96, 142, 143, 162, 164, 208, 209, 210, 239] or event/action/pose anticipation in videos [29, 66, 114, 192, 206, 240, 242]. However, these works focus on future prediction only on passive data (i.e., images or videos) and seldom apply the prediction for later planning. It is because they typically evaluate the model on a passive dataset without an agent who can actively interact with an environment. For example, among the works, the direct evaluation protocol heavily relies on predicting predefined event/action categories [134, 192, 240, 242], forecasting labeled pedestrian/human body/objects trajectories [96, 105, 143, 164, 171], or anticipating the timestamp when an event will happen [27, 196, 240]. On the contrary, an embodied agent can move in the environment, interact with the environment, and even change the environment. These interactions between the agent and the environment matter since future observations depend on the current plan. Likewise, the current observations are the results led by decision-making that happened in the past. Because of this active and feedback interaction between the agent and its visual observations, the agent's goal is not only to make a single correct prediction but also to make a sequence of predictions causally to accomplish a given task. In this vein, we believe that the visual forecasting ability can benefit the embodied agents to solve tasks more reliably.

1.2 Planning with Forecasting

In many fields, several approaches [18, 84, 95, 129, 144, 155, 172, 181, 207] have been proposed to perform planning based on future state rollouts. Nonetheless, these works engage in planning with perfect information. For instance, the agent state is captured by precise sensors in real-time [172, 207], and the agent's dynamic model is well-defined [144, 181]. In this way, the planner can obtain precise future states effortlessly. The emphasis becomes how to sample future information just enough or how to implement it in a real-world scenario [57, 129, 144, 155]. However, assuming that perfect sensors and dynamics models of the agent and environment are always available is unrealistic. For example, during the deployment stage, the sensors may encounter noise or even break [111, 146], the environment may exist in unseen and uncommon conditions [187, 214, 236], or the model of the interacting objects may be unavailable. Therefore, developing a planning algorithm based on perfect future state prediction is far from enough to successfully and responsibly deploy an embodied agent to the real world. In this regard, we are interested in embedding the visual forecasting model into an embodied policy, where (1) the policy only receives visual observations and a necessary task prompt (*i.e.*, object type for Object Navigation) rather than perfect information so that (2) the agent can make a future prediction on-the-fly and adjust its plan accordingly and simultaneously.

1.3 Policy Learning with Visual Forecasting: More Efficiently, More Effectively, and More Robustly

Predicting the future state of both agent and the surrounding environment before making a plan is essential. It becomes more crucial when the task has to be solved swiftly. For example, to catch a ball, a catcher agent has to adjust its velocity based on the ball's trajectory prediction. Without the trajectory prediction, the agent might move too far due to its movement inertia and fail the task. Likewise, the agent can better accomplish an interactive task if it can foresee the environment state-changes after an interaction. For instance, to push in a chair, a mobile manipulator agent can execute a precise amount of force in an accurate direction once it understands the outcome of that action. This way,

the agent can solve the task involving agent-environment interactions more effectively. Last but not least, the agent can adapt to unexpected situations if it can calibrate its belief about the effects of actions. Taking navigation as an example, the agent would assume it will move forward by 0.2m with a `MOVE` action. However, because the agent moves on a carpet with much larger friction than a regular wood floor, the actual movements become 0.1m. Therefore, this navigation agent should adapt its belief based on recent observations (e.g., moving on a carpet and the friction becomes more significant). Then, according to the updated belief (i.e., after applying `MOVE`, how far will the agent move?), the policy could choose the action leading the agent to a better state. In short, we argue that integrating the capability of applying visual forecasting for planning into an agent is the key to solving interactive embodiment tasks in three ways: more efficient planning, more effective decision-making, and more robust deployments.

In conclusion, we focus on bridging visual forecasting and model-free policy learning. In this way, the policy network can dynamically calibrate its plan according to the most recent forecasting results. Meanwhile, the visual forecasting model can also produce action-aware future predictions based on the plan sketched in the policy. Furthermore, the model framework is flexible, where the visual forecasting model and the policy network can be learned independently or trained end-to-end. Finally, we alleviate the perfect information assumption, where we learn the visual forecasting model via past experiences in the same episode led by the policy network.

1.4 Thesis Overview

In the following chapters, I will introduce my first project with the idea of visual forecasting for efficient planning (Visual Reaction) in Chapter 2. Then, I will share my second project using visual forecasting for effective interactive visual navigation, the Neural Interaction Engine, in Chapter 3. Furthermore, I will present my third work about robustly adapting a policy to unexpected action outcomes with the formulation of visual forecasting and the design of order-invariant policy in Chapter 4. Finally, I will summarize this thesis and share some thoughts regarding future directions in Chapter 5.

Publication

Portion of the thesis has been (or will be) appeared in the following publications:

◇ "Visual Reaction: Learning to Play Catch with Your Drone"

Kuo-Hao Zeng, Roozbeh Mottaghi, Luca Weihs, and Ali Farhadi (CVPR 2020)

◇ "Pushing it out of the Way: Interactive Visual Navigation"

Kuo-Hao Zeng, Luca Weihs, Ali Farhadi, and Roozbeh Mottaghi (CVPR 2021)

◇ "Moving Forward by Moving Backward: Embedding Action Impact over Action Semantics"

Kuo-Hao Zeng, Roozbeh Mottaghi, Luca Weihs, and Ali Farhadi (ICLR 2023, Oral)

Chapter 2

VISUAL REACTION: VISUAL FORECASTING FOR EFFICIENT PLANNING

In this chapter, we address the problem of *visual reaction*: the task of interacting with dynamic environments where the changes in the environment are not necessarily caused by the agent itself. Visual reaction entails predicting the future changes in a visual environment and planning accordingly. We study the problem of visual reaction in the context of playing catch with a drone in visually rich synthetic environments. This is a challenging problem since the agent is required to learn (1) how objects with different physical properties and shapes move, (2) what sequence of actions should be taken according to the prediction, (3) how to adjust the actions based on the visual feedback from the dynamic environment (e.g., when objects bouncing off a wall), and (4) how to reason and act with an unexpected state change in a timely manner. We propose a new dataset for this task, which includes 30K throws of 20 types of objects in different directions with different forces. Our results show that our model that integrates a forecaster with a planner outperforms a set of strong baselines that are based on tracking as well as pure model-based and model-free RL baselines. The code and dataset are available at prior.allenai.org/projects/visual-reaction.

2.1 Introduction

One of the key aspects of human cognition is the ability to interact and react in a visual environment. When we play tennis, we can predict how the ball moves and where it is supposed to hit the ground so we move the tennis racket accordingly. Or consider the scenario in which someone tosses the car keys in your direction and you quickly reposition your hands to catch them. These capabilities in humans start to develop during infancy and they are at the core of the cognition system [22, 37].

Visual reaction requires predicting the future followed by planning accordingly. The future prediction problem has received a lot of attention in the computer vision community.

The work in this domain can be divided into two major categories. The first category considers predicting future actions of people or trajectories of cars (e.g., [26, 105, 114, 211]). Typically, there are multiple correct solutions in these scenarios, and the outcome depends on the intention of the people. The second category is future prediction based on the physics of the scene (e.g., [117, 143, 218, 247]). The works in this category are mostly limited to learning from passive observation of images and videos, and there is no interaction or feedback involved during the prediction process.

In this capter, we tackle the problem of *visual reaction*: the task of predicting the future movements of objects in a dynamic environment and planning accordingly. The interaction enables us to make decisions on the fly and receive feedback from the environment to update our belief about the future movements. This is in contrast to passive approaches that perform prediction given pre-recorded images or videos. We study this problem in the context of playing catch with a drone, where the goal is to catch a thrown object using only visual ego-centric observations (Figure 4.1). Compared to the previous approaches, we not only need to predict future movements of the objects, but also to infer a minimal set of actions for the drone to catch the object in a timely manner.

This problem exhibits various challenges. First, objects have different weights, shapes and materials, which makes their trajectories different. Second, the trajectories vary based on the magnitude and angle of the force used for throwing. Third, the objects might collide with the wall or other structures in the scene, and suddenly change their trajectory. Fourth, the drone movements are not deterministic so the same action might result in different movements. Finally, the agent has limited time to reason and react to the dynamically evolving scene to catch the object before it hits the ground.

Our proposed solution is an adaptation of the model-based Reinforcement Learning paradigm. More specifically, we propose a forecasting network that rolls out the future trajectory of the thrown object from visual observation. We integrate the forecasting network with a model-based planner to estimate the best sequence of drone actions for catching the object. The planner is able to roll out sequences of actions for the drone using the dynamics model and an action sampler to select the best action at each time step. In other words, we learn a policy using the rollout of both object and agent movements.

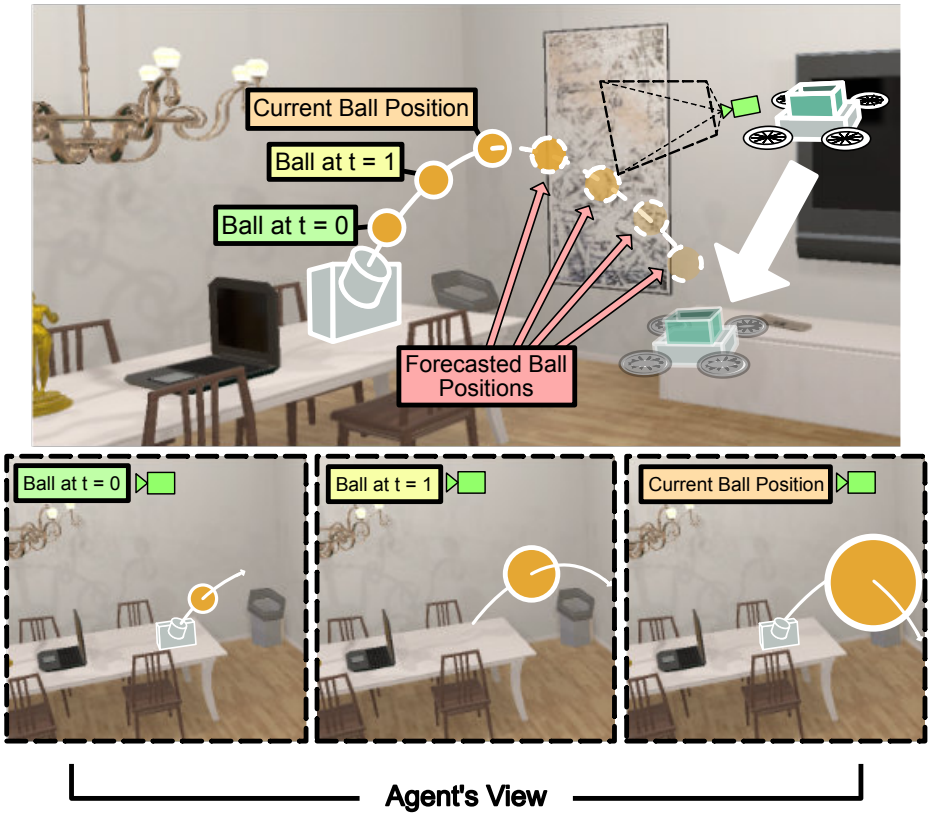

Figure 2.1: **Visual Reaction: Visual Forecasting for Efficient Planning.** Our goal is to train an agent that can visually react with an interactive scene. In the studied task, the environment can evolve independently of the agent. There is a launcher in the scene that throws an object with different force magnitudes and in different angles. The drone learns to predict the trajectory of the object from ego-centric observations and move to a position that can catch the object. The trajectory of the thrown objects varies according to their weight and shape and also the magnitude and angle of the force used for throwing.

We perform our experiments in AI2-THOR [106], a near-photo-realistic interactive environment which models physics of objects and scenes (object weights, friction, collision, etc). Our experiments show that the proposed model outperforms baselines that are based on tracking (current state estimation as opposed to forecasting) and also pure model-free and model-based baselines. We provide an ablation study of our model and show how the

performance varies with the number of rollouts and also the length of the planning horizon. Furthermore, we show how the model performs for object categories unseen during training.

2.2 Related Work

2.2.1 Future prediction & Forecasting.

Various works explore future prediction and forecasting from visual data. Several authors consider the problem of predicting the future trajectories of objects from individual [142, 162, 208, 209, 210, 239] and multiple sequential [6, 105, 229] images. Unlike these works, we control an agent that interacts with the environment which causes its observation and viewpoint to change over time. A number of approaches explore prediction from ego-centric views. [157] predict a plausible set of ego-motion trajectories. [171] propose an Inverse Reinforcement Learning approach to predict the behavior of a person wearing a camera. [206] learn visual representation from unlabelled video and use the representation for forecasting objects that appear in an ego-centric video. [116] predict the future trajectories of interacting objects in a driving scenario. Our agent also forecasts the future trajectory based on ego-centric views of objects, but the prediction is based on physical laws (as opposed to peoples intentions). The problem of predicting future actions or the 3D pose of humans has been explored by [29, 66, 114, 192]. Also, [26, 132, 188, 204, 205, 231] propose methods for generating future frames. Our task is different from the mentioned approaches as they use pre-recorded videos or images during training and inference, while we have an interactive setting. Methods such as [62] and [47] consider future prediction in interactive settings. However, [62] is based on a static third-person camera and [47] predicts the effect of agent actions and does not consider the physics of the scene.

2.2.2 Planning.

There is a large body of work (e.g., [33, 72, 80, 88, 148, 166, 186, 198, 215]) that involves a model-based planner. Our approach is similar to these approaches as we integrate the forecaster with a model-based planner. The work of [23] shares similarities with our approach. The authors propose learning a compact latent state-space model of the environment and

its dynamics; from this model an Imagination-Augmented Agent [166] learns to produce informative rollouts in the latent space which improve its policy. We instead consider visually complex scenarios in 3D so learning a compact generative model is not as straightforward. Also, [215] adopts a model-based planner for the task of vision and language navigation. They roll out the future states of the agent to form a model ensemble with model-free RL. Our task is quite different. Moreover, we consider the rollouts for both the agent and the moving object, which makes the problem more challenging.

2.2.3 Object catching in robotics.

The problem of catching objects has been studied in the robotics community. Quadrocopters have been used for juggling a ball [144], throwing and catching a ball [172], playing table tennis [184], and catching a flying ball [191]. [103] consider the problem of catching in-flight objects with uneven shapes. These approaches have one or multiple of the following issues: they use multiple external cameras and landmarks to localize the ball, bypass the vision problem by attaching a distinctive marker to the ball, use the same environment for training and testing, or assume a stationary agent. We acknowledge that experiments on real robots involve complexities such as dealing with air resistance and mechanical constraints that are less accurately modeled in our setting.

2.2.4 Visual navigation.

There are various works that address the problem of visual navigation towards a static target using deep reinforcement learning or imitation learning (e.g., [76, 136, 176, 232, 250]). Our problem can be considered as an extension of these works since our target is moving and our agent has a limited amount of time to reach the target. Our work is also different from drone navigation (e.g., [69, 174]) since we tackle the visual reaction problem.

2.2.5 Object tracking.

Our approach is different from object tracking (e.g., [17, 39, 59, 149, 193]) as we forecast the future object trajectories as opposed to the current location. Also, tracking methods

typically provide only the location of the object of interest in video frames and do not provide any mechanism for an agent to take actions.

2.3 Task definition

Our goal is to learn a policy to catch a thrown object using an agent that moves in 3D space. There is a launcher in the environment that throws objects in the air with different forces in different directions. The agent needs to predict the future trajectory of the object from the past observations (three consecutive RGB images) and take actions at each timestep to intercept the object. An episode is successful if the agent catches the object, i.e. the object lies within the agent's top-mounted basket, before the object reaches the ground. The trajectories of objects vary depending on their physical properties (e.g., weight, shape, and material). The object might also collide with walls, structures, or other objects, and suddenly change its trajectory.

For each episode, the agent and the launcher start at a random position in the environment (more details in Sec. 2.5). The agent must act quickly to reach the object in a short time before the object hits the floor or goes to rest. This necessitates the use of a forecaster module that should be integrated with the policy of the agent. We consider 20 different object categories, including *alarm clock*, *apple*, *basketball*, *book*, *bowl*, *bread*, *candle*, *cup*, *glass bottle*, *lettuce*, *mug*, *newspaper*, *salt shaker*, *soap bottle*, *statue*, *tissue box*, *toaster*, *toilet paper*, *vase* and *watering can*. The statistics about these 20 objects are shown in the Fig. 2.2, including the *mass*, *average acceleration* along the trajectories, *bounciness*, *drag*, and *angular drag*. Drag is the tendency of an object to slow down due to friction.

The model receives ego-centric RGB images from a camera that is mounted on top of the drone agent as input, and outputs an action $a_{d_t} = (\Delta_{v_x}, \Delta_{v_y}, \Delta_{v_z}) \in [-25m/s^2, 25m/s^2]^3$ for each timestep t, where, for example, Δ_{v_x} shows acceleration, in meters, along the x-axis. The movement of the agent is not deterministic due to the time dependent integration scheme of the physics engine. In the following, we denote the agent and object state by $s_d = [d, v_d, a_d, \phi, \theta]$ and $s_o = [o, v_o, a_o]$, respectively. d, v_d and a_d denote the position, velocity and acceleration of the drone and o, v_o and a_o denote those of the object. ϕ and θ specify the orientation of the agent camera, which can rotate independently from the agent.

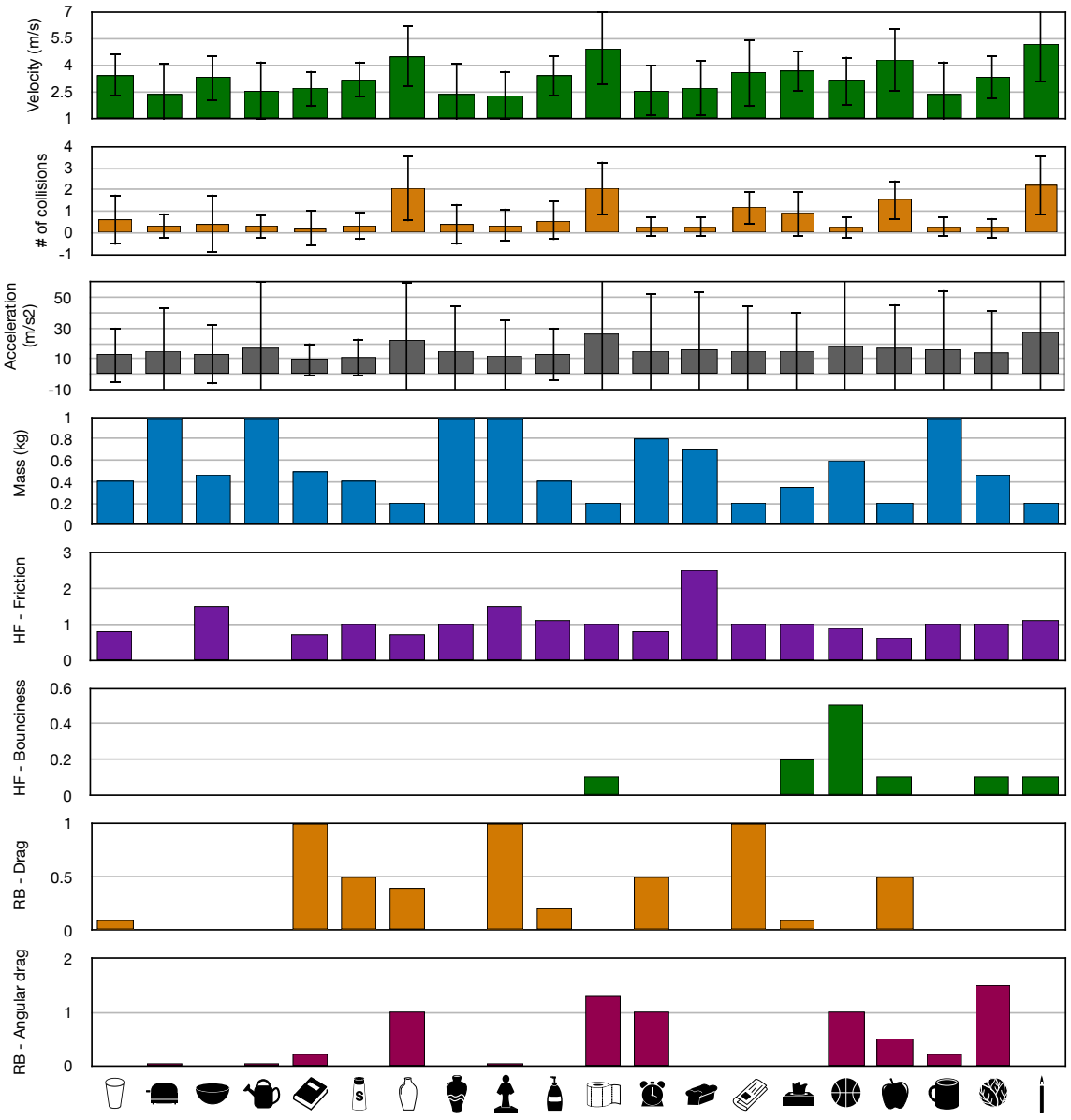

Figure 2.2: **More dataset statistics.** We provide the statistics for the 20 types of objects in our dataset. We illustrate the average velocity along the trajectories and the number of collisions with walls or other structures in the scene, the mass, average acceleration along the trajectories, bounciness, drag, and angular drag.

2.4 Model

2.4.1 Model Overview

Our model has two main components: a *forecaster* and a *model-predictive planner*, as illustrated in Fig. 2.3. The forecaster receives the visual observations $i_{t-2:t}$ and the estimated agent state s_{d_t} at time t, and predicts the current state s_{o_t} of the thrown object. The forecaster further uses the predicted object state (i.e., position, velocity and acceleration) to forecast H steps of object states $s_{o_{t+1:t+H}}$ in the future. The model-predictive planner is responsible for generating the best action for the agent such that it intercepts the thrown object. The model-predictive planner receives the future trajectory of the object from the forecaster and also the current estimate of the agent state as input and outputs the best action accordingly. The model-predictive planner includes an action sampler whose goal is to sample N sequences of actions given the current estimate of the agent state, the predicted object trajectory, and the intermediate representation r_t produced by the visual encoder in the forecaster. The action sampler samples actions according to a policy network that is learned. The second component of the model-predictive planner consists of a physics model and a model-predictive controller (MPC). The physics model follows Newton Motion Equation to estimate the next state of the agent (i.e., position and velocity at the next timestep) given the current state and action (that is generated by the action sampler). Our approach builds on related joint model-based and model-free RL ideas. However, instead of an ensemble of model-free and model-based RL for better decision making [215, 113], or using the dynamics model as a data augmentor/imaginer [166, 60] to help the training of model-free RL, we explicitly employ model-free RL to train an action sampler for the model-predictive planner.

In the following, we begin by introducing our forecaster, as shown in Fig. **??**(a), along with its training strategy. We then describe how we integrate the forecaster with the model-predictive planner, as presented in Fig. 2.3 and Fig. **??**(b). Finally, we explain how we utilize model-free RL to learn the action sampler used in our planner, Fig. **??**(b).

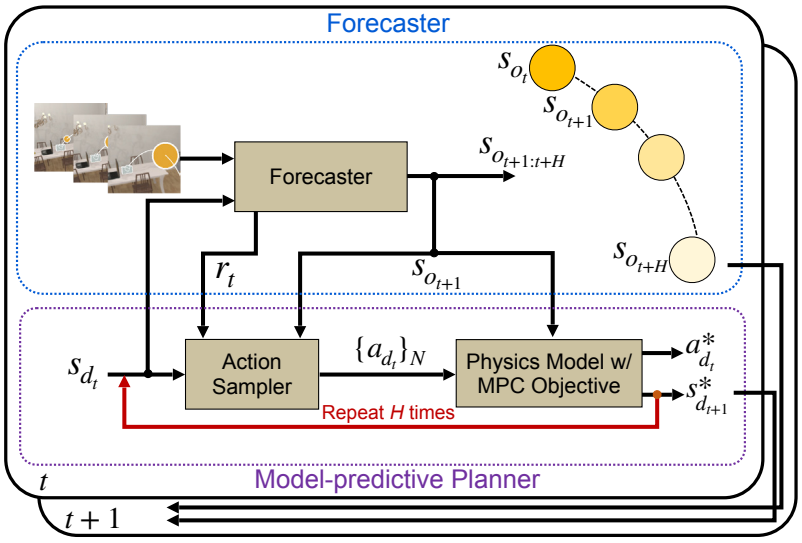

Figure 2.3: **Model overview.** Our model includes two main parts: Forecaster and Planner. The visual encoding of the frames, object state, agent state and action are denoted by r, s_o, s_d, and a, respectively. t denotes the timestep, and H is the planning horizon.

2.4.2 Forecaster

The purpose of the forecaster is to predict the current object state s_{o_t}, which includes the position $o_t \in \mathbb{R}^3$, the velocity $v_{o_t} \in \mathbb{R}^3$, and the acceleration $a_{o_t} \in \mathbb{R}^3$, and then, based on the prediction, forecast future object positions $o_{t+1:t+H}$ from the most recent three consecutive images $i_{t-2:t}$. The reason for forecasting H timesteps in the future is to enable the planner to employ MPC to select the best action for the task. Note that if the agent does not catch the object in the next timestep, we query the forecaster again to predict the trajectory of the object $o_{t+2:t+H+1}$ for the next H steps. Forecaster also produces the intermediate visual representation $r_t \in \mathbb{R}^{256}$, which is used by the action sampler. The details are illustrated in Fig. 2.4. We define the positions, velocity, and acceleration in the agent's coordinate frame at its starting position.

The three consecutive frames $i_{t-2:t}$ are passed through a convolutional neural network (CNN). The features of the images and the current estimate of the agent state s_{d_t} are combined using an MLP, resulting in an embedding r_t. Then, the current state of the

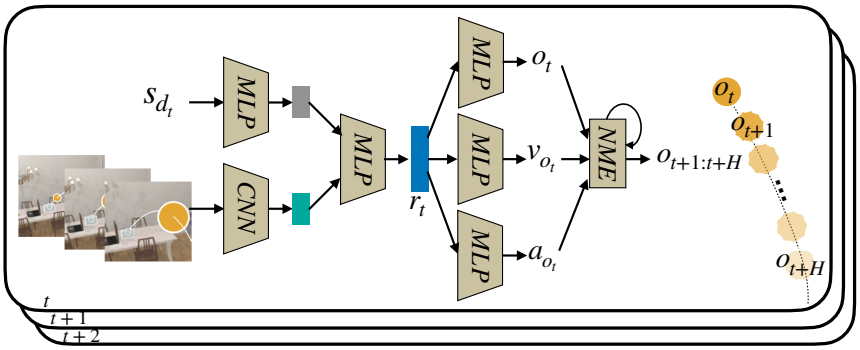

Figure 2.4: **Model architecture of the forecaster.** The forecaster receives images and an estimate of the agent state s_{d_t} as input and outputs the estimates for the current state s_{o_t}, including o_t, v_{o_t}, and a_{o_t}. Then it forecasts future positions of the object $o_{t+1:t+H}$ by discretized Newton Motion Equation. Forecasting is repeated every timestep if the object has not been caught.

object s_{o_t} is obtained from r_t through three separate MLPs. The NME, which follows the discretized Newton's Motion Equation ($o_{t+1} = o_t + v_t \times \Delta t$, $v_{t+1} = v_t + a_t \times \Delta t$) receives the predicted state of the object to calculate the future positions $o_{t+1:t+H}$. We take the derivative of NME and back-propagate the gradients through it in the training phase. Note that NME itself is not learned.

To train the forecaster, we provide the ground truth positions of the thrown object from the environment and obtain the velocity and acceleration by taking the derivative of the positions. We cast the position, velocity, and acceleration prediction as a regression problem and use the L1 loss for optimization.

2.4.3 Model-predictive Planner

Given the forecasted trajectory of the thrown object, our goal is to control the flying agent to catch the object. We integrate the model-predictive planner with model-free RL to explicitly incorporate the output of the forecaster.

Our proposed model-predictive planner consists of a model-predictive controller (MPC)

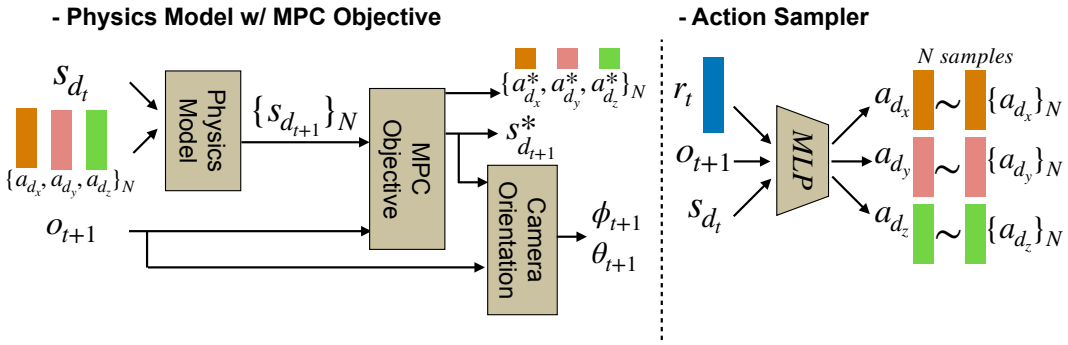

Figure 2.5: **Model architecture of the model-predictive planner.** The model-predictive planner includes a Model-Predictive Controller (MPC) w/ a Physics model and an action sampler. The action sampler generates N sequences $\mathbf{a}_{d_{t:t+H-1}} = \{(\Delta^j_{v_{x,i}}, \Delta^j_{v_{y,i}}, \Delta^j_{v_{z,i}})_{i=t}^{t+H-1}\}$, where $j = 1, ..., N$, of actions at each timestep, and an optimal action $(\Delta_{v_x^*}, \Delta_{v_y^*}, \Delta_{v_z^*})$ is chosen such that it minimizes the distance between the agent and the object at each timestep.

with a physics model, and an action sampler as illustrated in Fig. 2.5. We will describe how we design the action sampler in Sec. 2.4.4. The action sampler produces a rollout of future actions. The action is defined as the acceleration a_d of the agent. We sample N sequences of actions that are of length H from the action distribution. We denote these N sequences by $\mathbf{a}_{d_{t:t+H-1}}$. For each action in the N sequences, the physics model estimates the next state of the agent $s_{d_{t+1}}$ given the current state s_{d_t} by using the discretized Newton's Motion Equation ($d_{t+1} = d_t + v_{d_t} \times \Delta t$, $v_{d_{t+1}} = v_{d_t} + a_{d_t} \times \Delta t$). This results in N possible trajectories $d_{t+1:t+H}$ for the agent. Given the forecasted object trajectories $o_{t+1:t+H}$, the MPC then selects the best sequence of actions $\mathbf{a}^*_{t:t+H-1}$ based on the defined objective. The objective for MPC is to select a sequence of actions that minimizes the sum of the distances between the agent and the object over H timesteps. We select the first action a_t^* in the sequence of actions, and the agent executes this action. We feed in the agent's next state $s^*_{d_{t+1}}$ for planning in the next timestep.

Active camera viewpoint. The agent is equipped with a camera that rotates. The angle of the camera is denoted by ϕ and θ in the agent's state vector s_d. We use the estimated

object and agent position at time $t+1$, o_{t+1} and d^*_{t+1}, to compute the angle of the camera. We calculate the relative position $p \in (p_x, p_y, p_z)$ between object and agent by $o - d$. Then, we obtain the Euler angles along y-axis and x-axis by $\arctan \frac{p_x}{p_z}$ and $\arctan \frac{p_y}{p_z}$, respectively.

2.4.4 Action sampler

The actions can be sampled from a uniform distribution over the action space or a learned policy network. We take the latter approach and train a policy network which is conditioned on the forecasted object state, current agent state and visual representation. Model-based approaches need to sample a large set of actions at each timestep to achieve a high level of performance. To alleviate this issue, we parameterize our action sampler by a series of MLPs that learns an action distribution given the current agent state, the forecasted trajectory of the object $o_{t+1:t+H}$ and the visual representation r_t of observation $i_t - 2 : t$ (refer to Sec. 2.4.2). This helps to better shape the action distribution, which may result in requiring fewer samples and better performance.

To train our policy network, we utilize policy gradients with the actor-critic algorithm [194]. To provide the reward signal for the policy gradient, we use the 'success' signal (if the agent catches the object or not) as a reward. In practice, if the agent succeeds to catch the object before it hits the ground or goes to rest, it would receive a reward of +1. Furthermore, we also measure the distance between the agent trajectory and the object trajectory as an additional reward signal (pointwise distance at each timestep). As a result, the total reward for each episode is $R = \mathbb{1}\{\text{episode success}\} - 0.01 \cdot \sum_t ||d^*_t - o^*_t||_2$ where d^*_t and o^*_t are the ground truth positions of the agent and object at time t.

2.5 Experiments

2.5.1 Environment

We use AI2-THOR [106], which is an interactive 3D indoor virtual environment with near photo-realistic scenes. We use AI2-THOR v2.3.8, which implements physical properties such as object materials, elasticity of various materials, object mass and includes a drone agent. We add a launcher to the scenes that throws objects with random force magnitudes

in random directions.

The trajectories of the objects vary according to their mass, shape, and material. Sometimes the objects collide with walls or other objects in the scene, which causes sudden changes in the trajectory. Therefore, standard equations of motion are not sufficient to estimate the trajectories, and learning from visual data is necessary. The statistics of the average velocity and the number of collisions have been provided in Fig. 2.2.

The drone has a box on top to catch objects. The size of drone is $0.47m \times 0.37m$ with a height of $0.14m$, and the box is $0.3m \times 0.3m$ with a height of $0.2m$. The drone is equipped with a camera that is able to rotate. The maximum acceleration of the drone is $25m/s^2$ and the maximum velocity is $40m/s$. However, we provide results for different maximum acceleration of the drone. The action for the drone is specified by acceleration in x, y, and z directions. The action space is continuous, but is capped by the maximum acceleration and velocity.

2.5.2 Experiment settings

We use the *living room* scenes of AI2-THOR for our experiments (30 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. The drone and the launcher are assigned a random position at the beginning of every episode. We set the horizontal relative distance between the launcher and the drone to be 2 meters (any random position). We set the height of the launcher to be 1.8 meters from the ground which is similar to the average human height. The drone faces the launcher in the beginning of each episode so it observes that an object is being thrown.

To throw the object, the launcher randomly selects a force between $[40, 60]$ newtons, an elevation angle between $[45, 60]$ degree, and an azimuth angle between $[-30, 30]$ degree for each episode. The only input to our model at inference time is the ego-centric RGB image from the drone. We use 20 categories of objects such as basketball, alarm clock, and apple for our experiments. We observe different types of trajectories such as parabolic motion, bouncing off the walls and collision with other objects, resulting in sharp changes

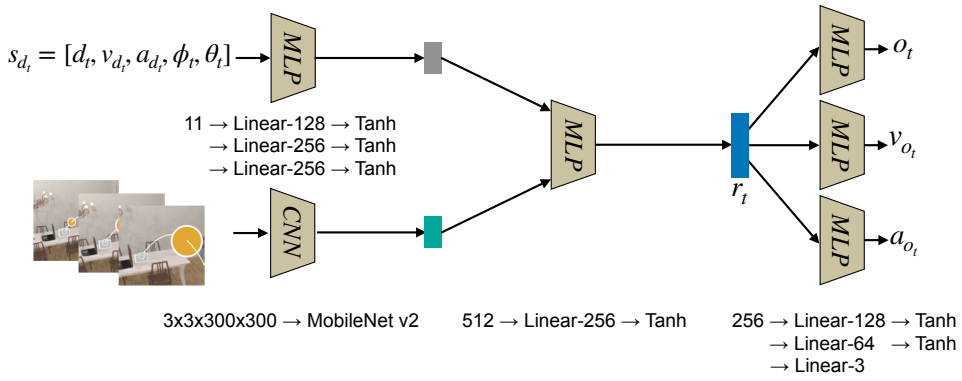

Figure 2.6: **Model details of the forecaster.** Detailed architecture of the forecaster, including layer type, number of parameters, and hidden feature dimensions.

in the direction. Note that each object category has different physical properties (mass, bounciness, etc.) so the trajectories are quite different. We use the same objects for training and testing. However, the scenes, the positions, the magnitude, and the angle of the throws vary at test time. We also show an experiment, where we test the model on categories unseen during training. We consider 20K trajectories during training, 5k for val and 5k for test. The number of trajectories is uniform across all object categories.

2.5.3 Implementation Details

We train our model by first training the forecaster. Then we freeze the parameters of the forecaster, and train the action sampler. An episode is successful if the agent catches the object. We end an episode if the agent succeeds in catching the object, the object falls on the ground, or the length of the episode exceeds 50 steps which is equal to 1 second. We use SGD with initial learning rate of 10^{-1} for forecaster training and decrease it by a factor of 10 every 1.5×10^4 iterations. For the policy network, we employ Adam optimizer [104] with a learning rate of 10^{-4}. We evaluate the framework every 10^3 iterations on the validation scenes and stop the training when the success rate saturates. We use MobileNet v2 [175], which is an efficient and light-weight network as our CNN model. The forecaster outputs the current object position, velocity, and acceleration. The action sampler provides a set

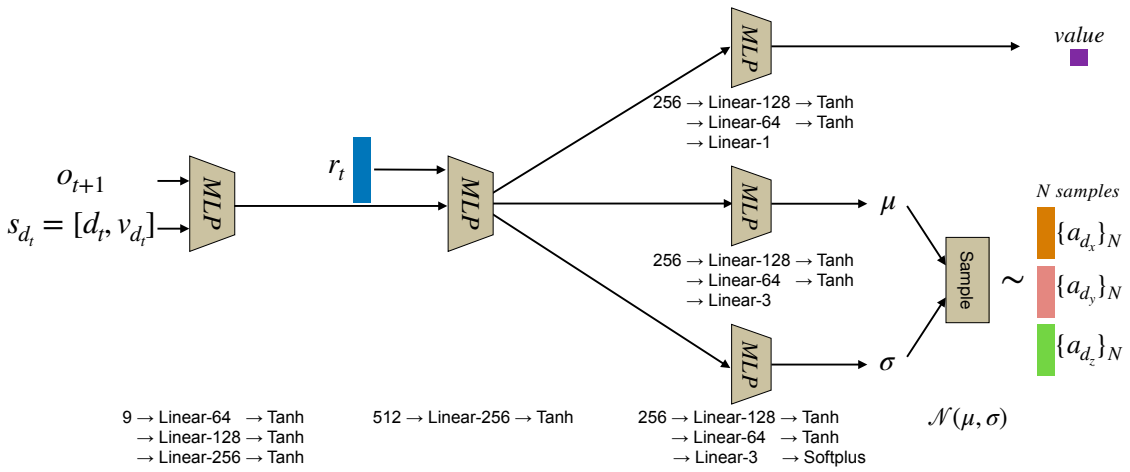

Figure 2.7: **Model details of the action sampler.** Detailed architecture action sampler, including layer type, number of parameters, and hidden feature dimensions.

of accelerations to the planner. They are all continuous numbers. Fig. 2.6 and Fig. 2.7 summarizes the details of the model architecture.

2.6 Baselines

2.6.1 Current Position Predictor (CPP).

This baseline predicts the current position of the object relative to the initial position of the drone in the 3D space, o_t, instead of forecasting the future trajectory. The model-predictive planner receives this predicted position at each time-step and outputs the best action for the drone accordingly. The prediction model is trained by an L1 loss with the same training strategy used for our method.

2.6.2 CPP + Kalman filter.

We implement this baseline by introducing the prediction update through time to the Current Position Predictor (CPP) baseline. We assume the change in the position of the object is linear and follows the Markov assumption in a small time period. Thus, we add the Kalman Filter [221] right after the output of the CPP. To get the transition probability, we

average the displacements along the three dimensions over all the trajectories in the training set. We set the process variance to the standard deviation of the average displacements, and measurement variance to 3×10^{-2}. Further, same as CPP, the model-predictive planner receives this predicted position at each time-step as input and outputs the best action to control the agent. This baseline is expected to be better than CPP, because the Kalman Filter takes into account the possible transitions obtained from the training set so it further smooths out the noisy estimations.

2.6.3 Model-free (A3C [138]).

Another baseline is model-free RL. We use A3C [138] as our model-free RL baseline. The network architecture for A3C includes the same CNN and MLP used in our forecaster and the action sampler. The network receives images $i_{t-2:t}$ as input and directly outputs action a_t for each time-step. We train A3C by 4 threads and use SharedAdam optimizer with the learning rate of 7×10^{-4}. We run the training for 8×10^4 iterations ($\approx$ 12 millions frames in total). In addition to using the the 'success' signal as the reward, we use the distance between the drone and the object as another reward signal.

2.7 Ablations

We use the training loss described in Sec. 2.4.2 and the training strategy mentioned in Sec. 2.5.3 for ablation studies.

2.7.1 Motion Equation (ME).

The forecaster predicts the position, velocity, and acceleration at the first time-step so we can directly apply motion equation to forecast all future positions. However, since our environment implements complex physical interactions, there are several different types of trajectories (e.g., bouncing or collision). We evaluate if simply using the motion equation is sufficient for capturing such complex behavior.

2.7.2 Uniform Action Sampling (AS).

In this ablation study, we replace our action sampler with a sampler that samples actions from a uniform distribution. This ablation shows the effectiveness of learning a sampler in our model.

Table 2.1: **Quantitative results.** We report the success rate for the baselines and the ablations of our model. N refers to the number of action sequences that the action sampler provides. The model-free baseline does not have an action sequence sampling component so we can provide only one number. The MPC upper bound is the case that model-predictive planner uses perfect forecasting with uniform action sampler. Note that the MPC upper bound must be done in the off-line mode since the perfect forecasting only available after collecting the objects' trajectory.

	N = 100k	**N = 10k**	**N = 1k**	**N = 100**	**N = 10**	**Best**
Curr. Pos. Predictor (CPP)	22.92±2.3	22.57±2.0	21.04±1.2	18.72±1.8	10.86±0.5	22.92±2.3
CPP + Kalman Filter	23.22±1.29	22.78±0.90	21.88±0.79	19.29±0.81	12.17±1.2	23.22±1.29
Model-free (A3C [138])	-	-	-	-	-	4.54±2.3
Ours, ME, uniform AS	6.12±0.7	6.11±0.7	6.00±0.5	5.99±0.5	5.12±1.0	6.12±0.7
Ours, uniform AS	26.01±1.3	25.47±1.3	23.61±1.5	20.65±0.93	10.58±1.1	26.01±1.3
Ours, full	**29.34**±0.9	**29.26**±1.4	**29.12**±0.8	**29.14**±0.8	**24.72**±1.6	**29.34**±0.9
MPC Upper bound	68.67±1.9	76.00±0.0	78.67±1.9	66.00±3.3	49.33±10.5	78.67±1.9

2.8 Results

In this section, we provide comparisons between our method and set of baselines that are described below. We also provide an ablation study of our model. We will illustrate qualitative examples of success and failure cases as well.

2.8.1 Quantitative results.

The results are summarized in Tab. 2.1 for all 20 objects and different number of action sequences. We use success rate as our evaluation metric. Recall that the action sampler

samples N sequences of future actions. We report results for five different values $N = 10, 100, 1\text{k}, 10\text{k}, 100\text{k}$. We set the horizon H to 3 for the forecaster and the planner. For evaluation on the test set, we consider 5k episodes for each model. For Tab. 2.1, we repeat the experiments 3 times and report the average.

As shown in the table, both the current position predictors (CPP) and the Kalman Filter (CPP + Kalman Filter) baseline are outperformed by our model, which shows the effectiveness of forecasting compared to estimating the current position. Our full method outperforms the model-free baseline, which shows the model-based portion of the model helps improving the performance. 'Ours, ME, uniform AS' is worse than the two other variations of our method. This shows that simply applying motion equation and ignoring complex physical interactions is insufficient and it confirms that learning from visual data is necessary. We also show that sampling from a learned policy 'Ours - full' outperforms 'Ours, uniform AS', which samples from a uniform distribution. This justifies using a learned action sampler and shows the effectiveness of the integration of model-free and model-based learning by the model-predictive planner.

2.8.2 Qualitative results

Fig. 2.8 shows two sequences of catching the object and a failure case. The sequence is shown from a third person's view and the agent camera view (we only use the camera view as the input to our model). The second row shows the drone is still able to catch the object although there is a sudden change in the direction due to the collision of the object with the ceiling. A supplementary video at https://youtu.be/iyAoPuHxvYs shows more success and failure cases.

2.9 Analysis

2.9.1 Per-category results.

Tab. 2.2 shows the results for each category for 'Ours - full' and 'Ours, uniform AS'. The results show that our model performs better on relatively heavy objects. This is expected since typically there is less variation in the trajectories of heavy objects.

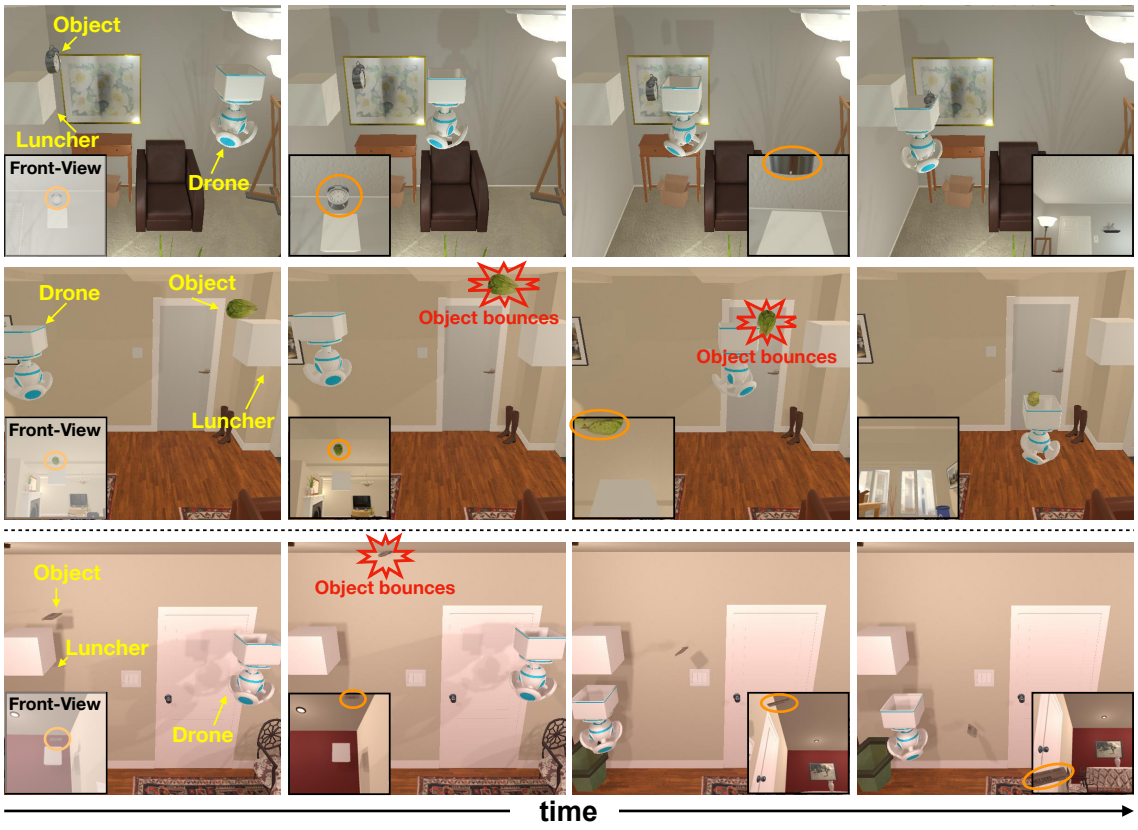

Figure 2.8: **Qualitative Results.** We show two successful sequences of catching objects in the first two rows and a failure case in the third row. For instance, in the second row, the object bounces off the ceiling, but the drone is still able to catch it.

2.9.2 Difficulty-based categorization.

Tab. 2.3 shows the performance achieved by 'Ours - full' and 'Ours, uniform AS' in terms of difficulty of the trajectory. The difficulty is defined by how many times the object collides with other structures before reaching the ground or being caught by the agent. We define *easy* by no collision, *medium* by colliding once, and *difficult* by more than one collision. The result shows that even though our model outperforms baselines significantly, it is still not as effective for *medium* and *difficult* trajectories. It suggests that focusing on modeling more complex physical interactions is important for future research.

Table 2.2: **Per category result.** Our dataset includes 20 object categories. We provide the success rate for each object category.

Ours, uniform AS	20.4	49.8	32.4	13.7	39.2	17.3	12.9	6.0	37.6	26.8
Ours, full	22.8	65.9	35.2	20.1	37.2	18.5	14.5	12.0	42.8	29.6
Ours, uniform AS	0.0	49.6	34.8	24.0	25.6	61.2	14.0	12.8	39.6	8.4
Ours, full	0.0	54.4	37.2	24.4	26.4	64.4	18.1	18.0	40.8	10.0

Table 2.3: **Difficulty categorization.** We show the categorization of the results for different levels of difficulty.

	Easy	**Medium**	**Difficult**
Proportion	43%	33%	24%
Ours, uniform AS	46.4	18.4	1.2
Ours, full	51.9	20.7	1.6

2.9.3 Different mobility.

We evaluate how varying the mobility of the drone affects the performance (Tab. 2.4). We define the mobility as the maximum acceleration of the drone. We re-train the model using 100%, 80%, 60%, 40%, 20% of the maximum acceleration. The results show that the larger mobility results in better performance. This is expected since larger mobility increases the chance of the drone to visit more locations in the environment.

2.9.4 Movement noise

Here, we evaluate the scenarios where the agent has more noisy movements. We perform this by injecting a Gaussian noise to the drone's movement after each action (Fig. 2.9). We re-train the model using 0.01, 0.05, 0.1, and 0.15 of the standard deviation of the Gaussian noise. As expected, the performance decreases with more noise.

Table 2.4: **Mobility results.** We show the results using 100%, 80%, 60%, 40%, 20% of the maximum acceleration.

	100%	**80%**	**60%**	**40%**	**20%**
Ours, uniform AS	26.0	23.6	16.0	10.5	3.3
Ours, full	29.3	25.1	18.4	10.5	3.5

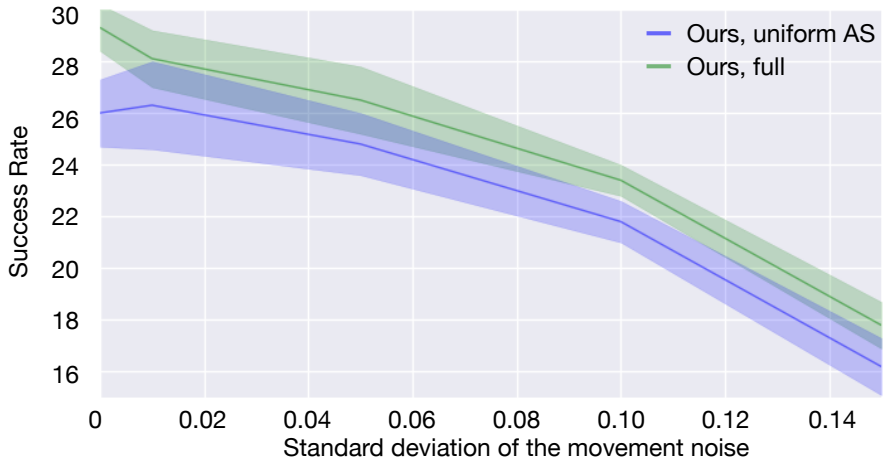

Figure 2.9: **Movement noise variation results.** We show how the noise in the agent movement affects the performance.

2.9.5 Results on unseen categories

We train the best model on 15 object categories and evaluate on the remaining categories[1]. The success rate is $29.12 \pm 0.9\%$. This shows that the model is rather robust to unseen categories.

[1]We selected a subset of 5 objects as our held-out set such that they have different physical properties: *basketball*, *bowl*, *bread*, *candle*, *watering can*. We trained our model on the rest of the objects: *alarm clock*, *apple*, *book*, *cup*, *glass bottle*, *lettuce*, *mug*, *newspaper*, *salt shaker*, *soap bottle*, *statue*, *tissue box*, *toaster*, *toilet paper* and *vase*.

2.9.6 Results for the case that the camera is fixed

In Tab. 2.5, we provide the results for the case that the drone camera is fixed and does not rotate. In this experiment, we set horizon $H = 3$ and number of action sequences $N = 100$k. The performance degrades for the case that the camera does not rotate, which is expected.

Table 2.5: **Camera orientation results.** We show the results for the scenario that the camera orientation does not change. **GT** corresponds to the case that we use the ground truth camera orientation at train/test time. The ground truth camera orientation is obtained via ground truth object's position and drone's position. **Est** denotes the case that we use the predicted object and drone positions to calculate to estimate the camera angle. **Fixed** denotes the case that the camera orientation is fixed.

	GT.	**Est.**	**Fixed**
Ours, uniform AS	54.7	26.0	18.6
Ours, full	59.3	29.3	20.2

2.9.7 Error of position, velocity and acceleration prediction.

The error of our method (L2 distance) for predicting position, velocity and acceleration are 0.644 ± 0.389, 0.037 ± 0.017, and 0.007 ± 0.014, respectively. The error for the baseline CPP for example is 0.686 ± 0.362, 0.148 ± 0.033, and 0.007 ± 0.013 for position, velocity and acceleration, respectively. This comparison is performed for $N = 100$k.

2.9.8 Different horizon length

Here, we show how the performance changes with varying the horizon length H (Fig. 2.10). We observe a performance decrease for horizons longer than 3. The reason is that the learned forecaster has a small error and the error for each time-step accumulates. Thus, training an effective model with longer horizons is challenging and we leave it for future research.

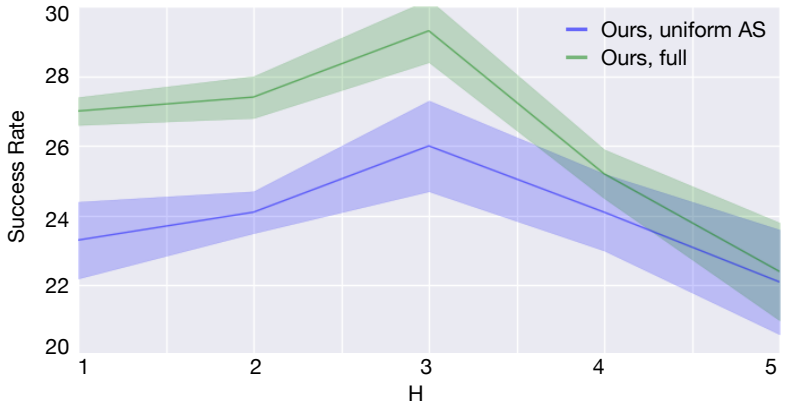

Figure 2.10: **Result for different horizon lengths.** We show how the performance changes by varying H.

2.9.9 Visualization of forecasted trajectory

We visualize two examples of the forecasted trajectory in Figure 2.11. The examples demonstrate that the drone is able to plan according to the future positions of the objects predicted by the forecaster.

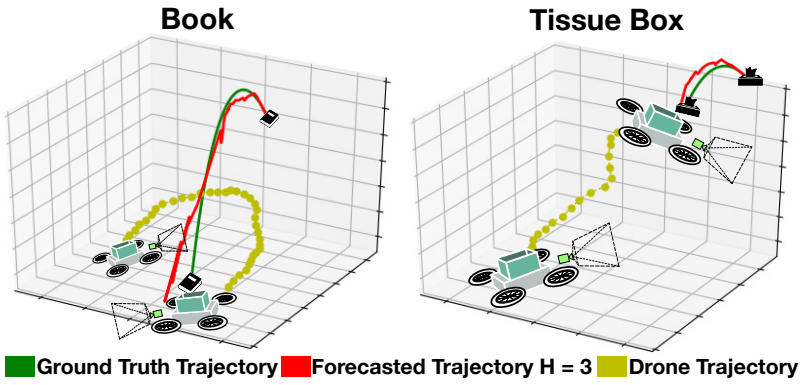

Figure 2.11: **Qualitative examples of the forecasting trajectory.** The green color, red color, and yellow color denote the ground truth object's trajectory, forecasted object's trajectory, and drone's moving trajectory, respectively.

2.10 Conclusion

We address the problem of *visual reaction* in an interactive and dynamic environment in the context of learning to play catch with a drone. This requies learning to forecast the trajectory of the object and to estimate a sequence of actions to intercept the object before it hits the ground. We propose a new dataset for this task, which is built upon the AI2-THOR framework. We showed that the proposed solution outperforms various baselines and ablations of the model including the variations that do not use forecasting, or do not learn a policy based on the forecasting.

Chapter 3

INTERACTIVE VISUAL NAVIGATION: VISUAL FORECASTING FOR EFFECTIVE INTERACTION

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 chapter, 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. The code and dataset are available at prior.allenai.org/projects/interactive-visual-navigation.

3.1 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 [8, 16, 30, 177, 223, 250], 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

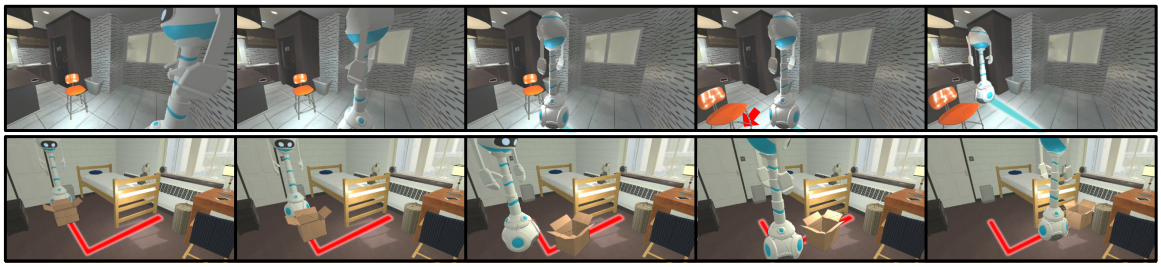

Figure 3.1: **Interactive Visual Navigation: Visual Forecasting for Effective Interaction.** Visual navigation may require interactions that go beyond moving forward or backward, and turning left or right. For example, the agent in the top row needs to push the chair out of its way to reach the target. Interactive navigation entails deeper understanding of the outcome of agents actions on objects in the scene. In this chapter, 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.

by an obstacle (e.g., a chair) as shown in Fig. 3.1 (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 chapter, 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 [107] 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.

3.2 Related Work

3.2.1 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 [25] segment a point cloud into salient regions and predict the rigid body motion. Li *et al.* [123] learn to re-position and re-orient an object with unknown physical properties. Several works [51, 52, 63, 241] 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 [230], learning forward and inverse scene dynamics from object interaction data [150], representing scenes as object-centric graphs and learning to predict changes in object pose after applying a push action [159], learning the dynamics of balls and walls in the game of billiards [65], and modeling the dynamics of robot interactions by jointly estimating forward and inverse models of dynamics [3]. 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.

3.2.2 Learning dynamics from perception.

The dynamics of objects can be inferred from images and videos alone without any interaction. [203] 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. [126] reason about the underlying physical properties of objects that appear in a sequence of frames and predict future motion of those objects. [99, 225] jointly train a perception module, an object-based physics engine and a renderer to generate the future predictions. [28] propose Neural Physics Engine that outputs the future states of objects and their properties. [217] also infers the physical state of objects from video input and predict their future trajectories. [224] infer physical properties of objects such as mass and density from videos. [142, 143, 241, 242] 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.

3.2.3 Visual navigation.

The tasks that we consider in this chapter involves visual navigation. Visual navigation has been addressed in various papers in recent Embodied AI literature. Most works focus on point navigation (PointNav) [8, 31, 177, 222] or object navigation (ObjectNav) [16, 30, 49, 223]. 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 [15,

100, 137, 189, 190, 227, 228]. The difference with [189, 190] is that those works are not learning-based and generalization to unseen scenes is not evaluated. Our task differs from that of [228] 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 [228].

3.3 Model

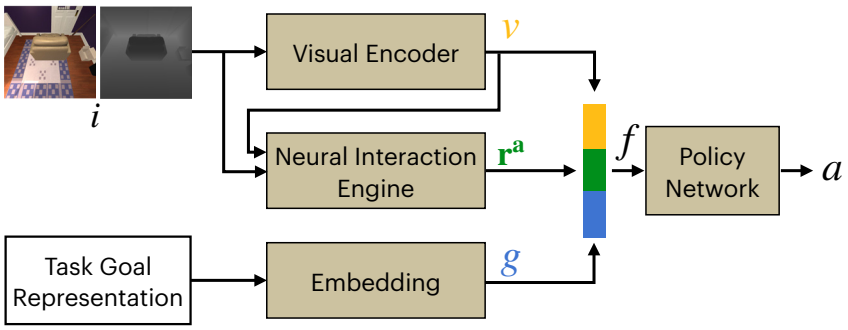

Figure 3.2: **Model overview.** Our model includes three main parts: Visual Encoder, Neural Interaction Engine, and Policy Network.

3.3.1 Model Overview

Our model has three main components: a visual encoder, Neural Interaction Engine, and a policy network, as illustrated in Fig. 3.2. 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. 3.3 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

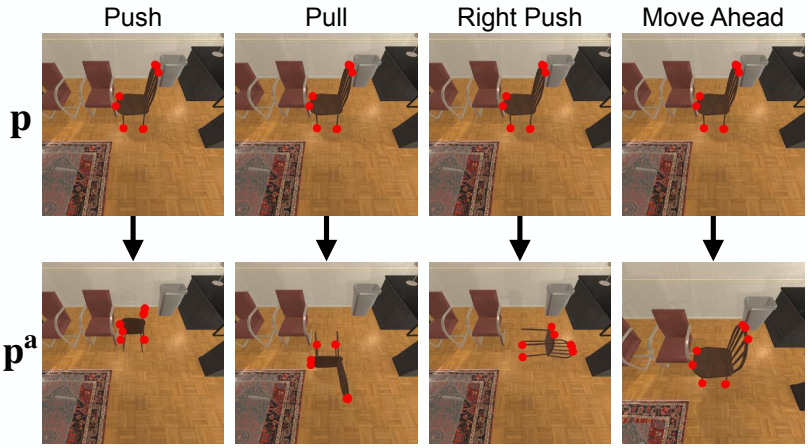

Figure 3.3: **Keypoint examples.** The top row shows object keypoints $\mathbf{p}_o$ and bottom row shows action-conditioned keypoints $\mathbf{p}_o^a$ resulted from `Push`, `Pull`, `RightPush` and `MoveAhead` actions. The keypoints are showon in red.

$m_o^a \in \mathbb{R}^{4\times4}$ 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.

3.3.2 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.

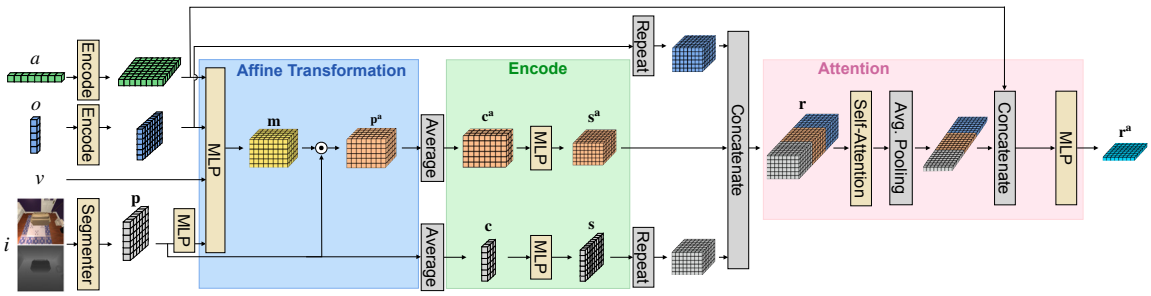

Figure 3.4: **Neural Interaction Engine.** The inputs to the neural interaction engine are action indices, object categories, visual representation v from the visual encoder, and visual observation i, which includes an RGB image and a depth map. After encoding each input modality, the engine uses an 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 $\mathbf{r}^a$.

As shown in Fig. 3.4, 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 [85] 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. 3.5.2 for more details). We used a heuristic approach to find the keypoints, but any other keypoint detection approach (e.g., [110, 195]) 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. 3.4. 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 $\mathbf{m} \in \mathbb{R}^{O \times |\mathcal{A}| \times 4 \times 4}$ 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 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. 3.4.

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 [200] 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. 3.4. 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.

3.3.3 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 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

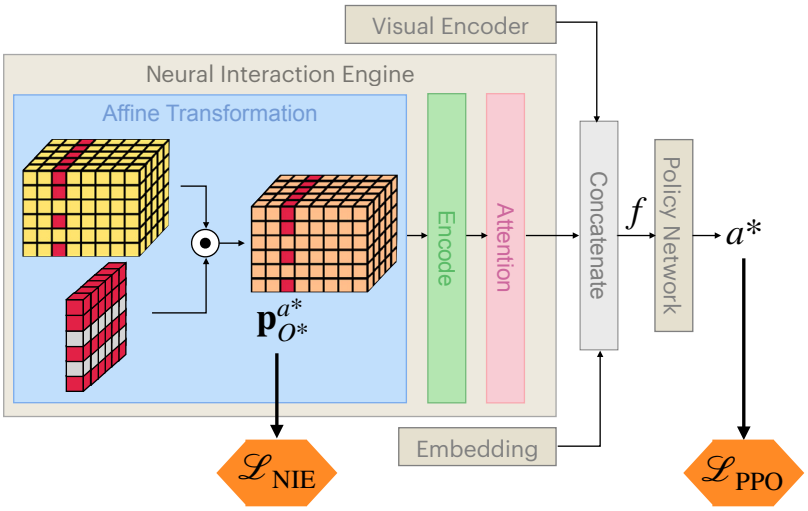

Figure 3.5: **Training pipeline.** The entire model is trained by $\mathcal{L}_{\text{PPO}}$ and the Affine Transformation module is trained by $\mathcal{L}_{\text{NIE}}$. However, the gradients backpropagated from $\mathcal{L}_{\text{NIE}}$ are only used to update the parameters corresponding to $\mathbf{p}^{a^*}_{O^*}$, 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.

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. 3.5), 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 $O^* \subset O$:

$$\mathcal{L}_{\text{NIE}} = L1(\mathbf{p}^{a^*}_{O^*}, \mathbf{t}^{a^*}_{O^*}). \tag{3.1}$$

Further, to learn the policy network, we employ the Proximal Policy Optimization (PPO) [179] to perform an on-policy reinforcement learning, as illustrated in Fig. 3.5. 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.

3.4 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. 3.5. In Sec. 3.6 and Sec. 3.7, we introduce our comparative baselines and variations of our model. Finally, we present quantitative and qualitative results in Sec. 3.8.

3.4.1 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 [8] 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.

3.4.2 Environment settings.

In this chapter, we perform experiments on AI2-iTHOR [107] 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`,

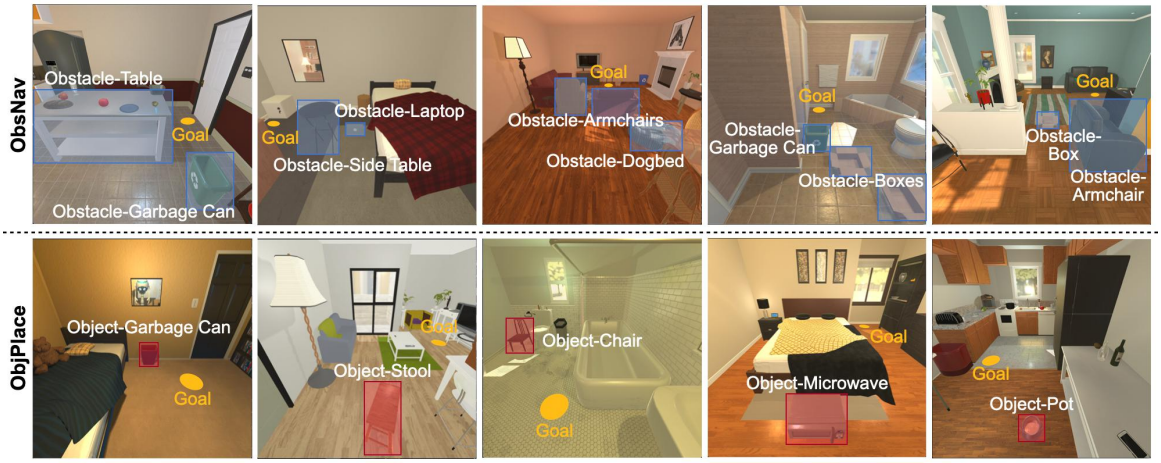

Figure 3.6: **Dataset examples.** Top: five examples in *ObsNav* dataset, where the blue boxes are obstacles and the yellow circle is the target position. Bottom: five examples in *ObjPlace* dataset, where the red boxes are the object that should be displaced and the yellow circle is the target place.

`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. 3.3 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.

3.4.3 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, including *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*. 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. 3.6 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. 3.6 shows five examples in this dataset.

3.5 Implementation Details

3.5.1 Learning details

In this chapter, we use the AllenAct [220] 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 introduced in Sec. 3.3.3 to 3. We discuss the effect of α on the performance in Sec. 3.8. 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.

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 [85] 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 [226] to pre-train the MaskRCNN (ResNet-50 with FPN) on our training scenes with 8k images for 10 epochs.

3.5.2 Heuristic corner detector

Fig. 3.7 (a) shows our heuristic keypoints detector pipeline. More specifically, we first use an object segmentation model to obtain the segmentation M corresponding to object $o =$ `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. 3.5.3) to obtain the object segmentation. Further we present the details of our heuristic corner detector in Fig. 3.7 (b), where the 8 corner points are obtained by 8 different criteria and each of the points has to be inside the segmentation M:

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

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. 3.8.

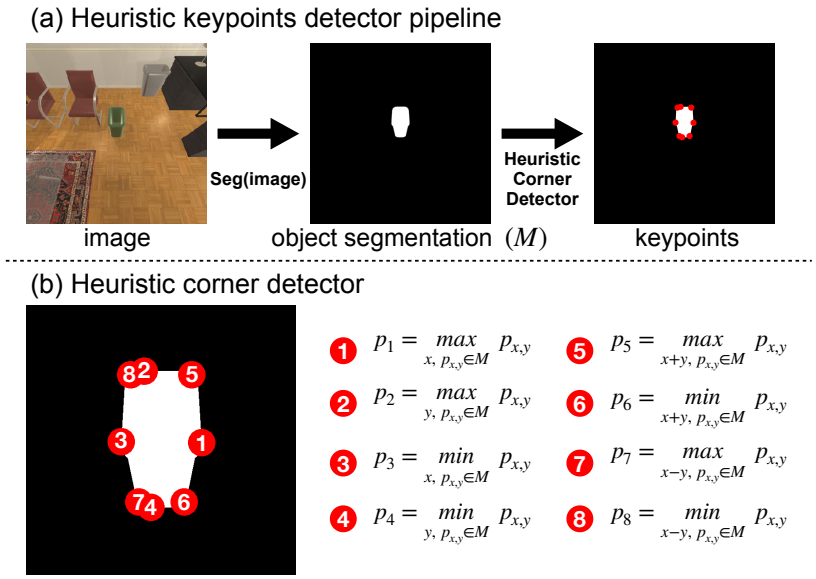

Figure 3.7: **Keypoint detector details.** (a) Heuristic keypoint detector pipeline. (b) Heurisitc corner detector.

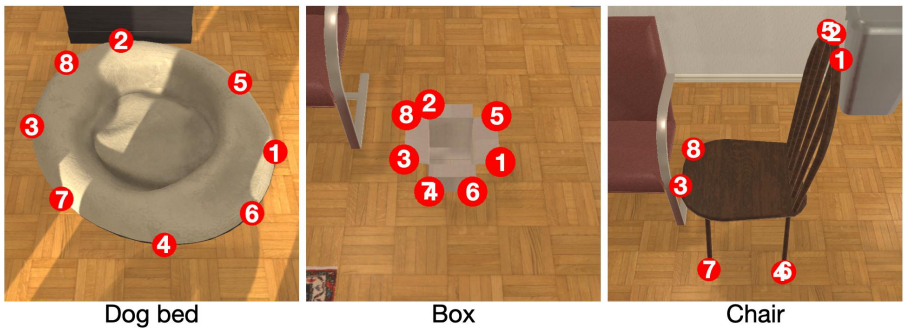

Figure 3.8: **Keypoints examples.** Examples keypoints obtained by our keypoint detector.

3.5.3 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. 3.9 shows qualitative results on 20 used objects in one of the testing scenes `LivingRoom227`.

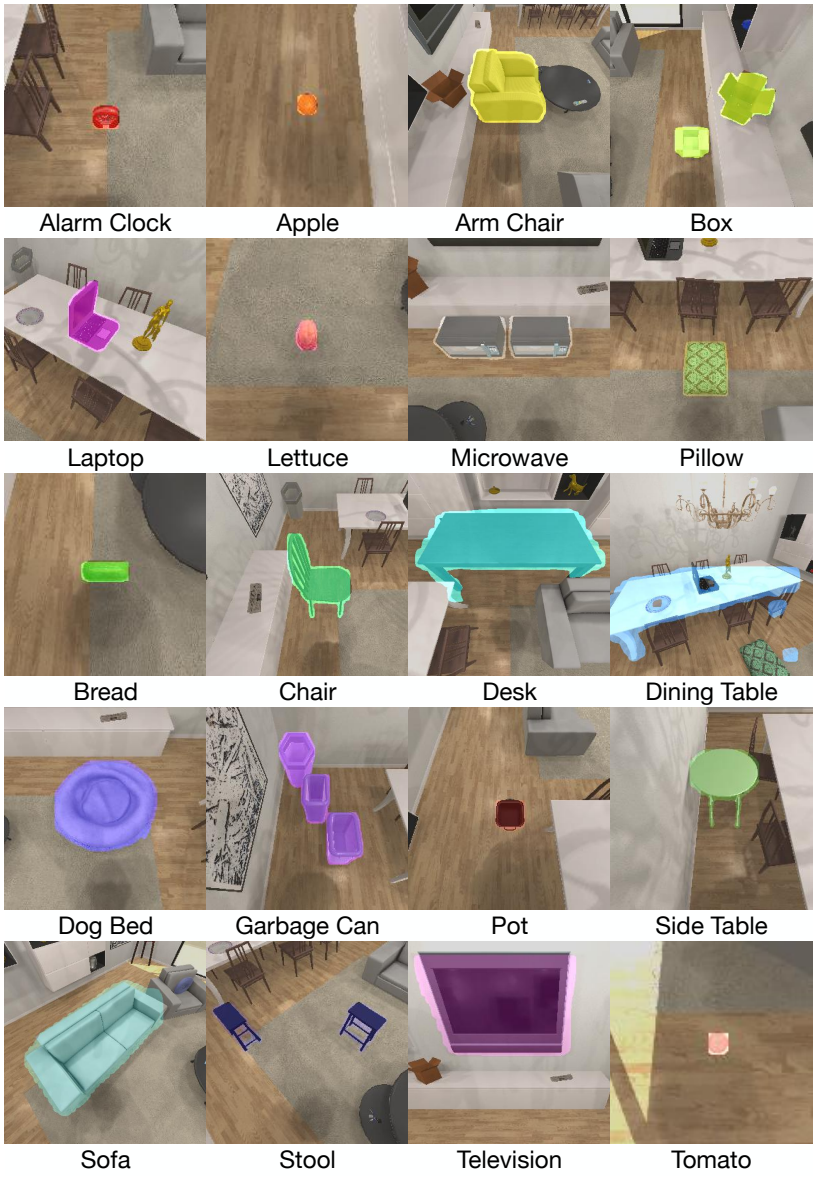

Figure 3.9: **MaskRCNN's qualitative results on 20 used objects.** We randomly spawn 20 objects in the testing scene `LivingRoom227` and apply the pretrained MaskRCNN to obtain the segmentation results. The object prediction score is set to 0.5 and the segmentation probability is set to 0.1.

3.5.4 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. 3.4 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). Fig.3.10 and Fig. 3.11 summarize the details of the architecture for visual encoder, goal embedding, policy network, and NIE model.

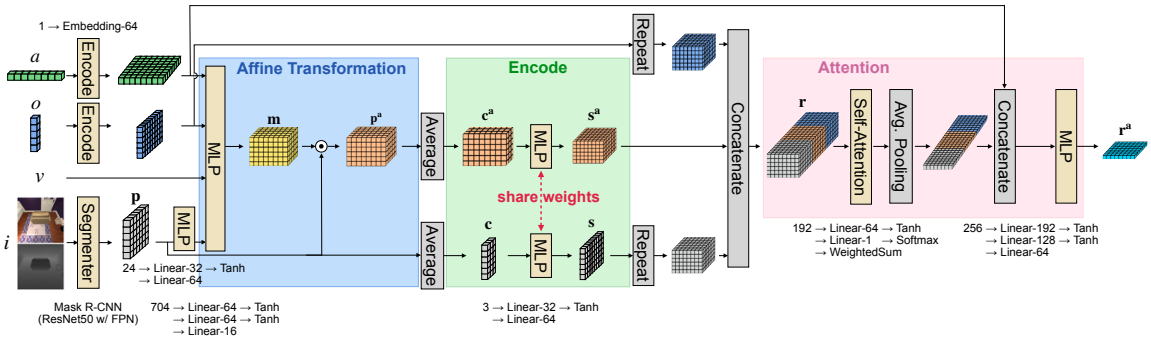

Figure 3.10: **Detailed architecture of the NIE model.** We show model details and hidden feature dimensions of the NIE model, including the Affine Transformation Module, the Encode Module, and the Attention Module.

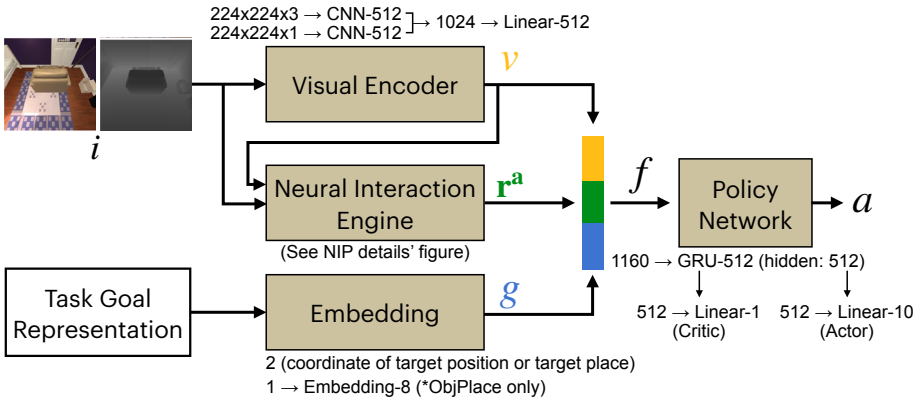

Figure 3.11: **Detailed architecture of the visual encoder, goal embedding, and policy network.** We show model details and hidden feature dimensions of the visual encoder, the goal embedding, and the policy network.

3.5.5 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 [151] important to learn the policy in the two studied tasks. We implement reward shaping for each task as follows:

• *ObsNav*: Similar to [177], 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.

3.6 Baselines

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

3.6.1 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 [179] and we use the same learning hyparameters mentioned in Sec. 3.5 to train using this method.

3.6.2 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. 3.5, we use the ground truth segmentation during the training stage and the results generated by MaskRCNN (fine-tuned on our data) at evaluation.

3.6.3 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. 3.5.2) 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.

3.6.4 PPO + auxiliary loss

We implement two baselines based on PBL [74] and CPC|A [73] to facilitate the policy learning upon the PPO baseline. During the training stage, both the PBL and 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.

3.7 Ablations

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

3.7.1 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. 3.5 to train this model.

3.7.2 NIE w/ $1 \times \mathcal{L}_{NIE}$

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.

3.7.3 NIE w/ $10 \times \mathcal{L}_{NIE}$

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.

3.8 Results

3.8.1 Evaluation Metrics

We evaluate all models by *Success Rate (SR)*, *Final Distance to Target (FDT)*, and *Success weighted by Path Length (SPL)* [7] 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.

3.8.2 ObsNav

The quantitative results of the *ObsNav* task are shown in Tab. 3.1. 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 the object will not move. It is not feasible

Table 3.1: **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.

Methods	**SR (%)** ↑	**FDT** (m) ↓	**SPL** ↑
Baselines:			
PPO [179]	67.1	0.605	25.7
RGB-D-S	62.8	0.499	25.0
RGB-D-K	70.9	0.459	25.8
PBL [74]	72.2	0.421	28.7
CPC|A [73]	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

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.

3.8.3 ObjPlace

The results are shown in Table 3.2. 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.

Table 3.2: **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.

Methods	**SR (%)** ↑	**FDT** (m) ↓	**SPL** ↑
Baselines:			
PPO [179]	1.2	3.18	0.85
RGB-D-S	1.2	3.15	0.85
RGB-D-K	1.3	2.84	0.88
PBL [74]	1.3	3.12	0.81
CPC|A [73]	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

3.8.4 Compare to PPO + auxiliary loss.

As mentioned in Sec. 3.6.4, we implemented PBL [74] and CPC|A [73] to compare our proposed NIE to the policy learned with auxiliary loss. Note that to have a fair comparison, we use GRU unit with the same hidden size and only predict one time step further in these two baselines. As shown in Tab. 3.1 as well as Tab. 3.2, the PBL achieves 77.5% and 0.5% (success rate) on ObsNav and ObjPlace, respectively, and the CPC|A achieves 79.3% and 12.1% (success rate) on ObsNav and ObjPlace, respectively. Our method outperforms both of these approaches on both tasks. It shows that using a simple contrastive predictive loss to predict agent's state as an auxiliary loss is not enough to effectively address the studied tasks. In contrast, our proposed NIE is able to capture the effects of interactions, thus achieves better performances.

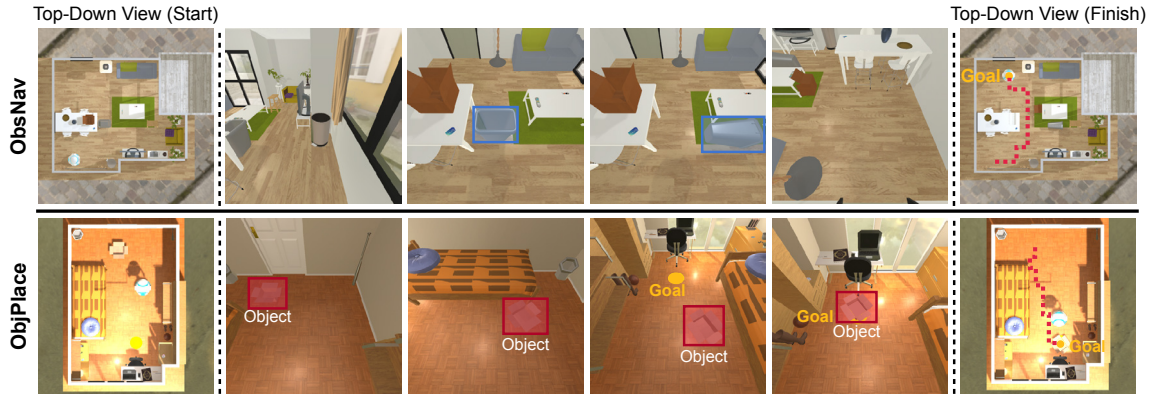

Figure 3.12: **Qualitative results.** Top: An example of the *ObsNav* task is shown. The blue box is the obstacle the agent should move away to unblock the path (the blue marking is just for visualization purposes and not visible to the agent). The agent's movement is shown by a dashed trajectory in 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.

3.8.5 Qualitative Results

We show qualitative results in Fig. 3.12. 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 at https://youtu.be/q8xqxgnLEY4 to show more successful and failure cases. We show how well the NIE model predicts the future location of keypoints conditioned on the actions.

3.9 Analysis

3.9.1 Estimate of the difficulty of two tasks.

The SPL metric that we report includes the episode length as well. In Tab. 3.3, we show results for Easy, Medium, and Hard cases. We found our model achieves better performance

on simpler tasks.

Table 3.3: **Difficulty categorization.** (Success and average # of steps) for different levels of difficulty. Easy, Medium and Hard (33, 33-66, 66+ percentile of the dataset in terms episode length).

	Easy	**Medium**	**Difficult**
ObsNav	84.2% (43.3)	82.2% (56.7)	80.3% (83.3)
ObjPlace	26.6% (32.9)	21.7% (44.0)	15.8% (55.4)

3.9.2 Improvement with self-attention.

We try our best model (NIE w/ $3\times\mathcal{L}_{NIE}$) with $1, 2, 5$ self-attention layers on ObjPlace. The results are in Tab. 3.4. We conjecture that the models with more self-attention layers may need more training data/iterations to achieve better results.

Table 3.4: **Different number of self-attention layers.** We show success rate on ObjPlace task using different number of self-attention layers.

	1 layer	**2 layers**	**5 layers**
ObjPlace	17.5%	20.7%	16.7%

3.9.3 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 $\mathcal{L}1$ 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. 3.13 shows the qualitative results of our action-conditioned keypoint prediction.

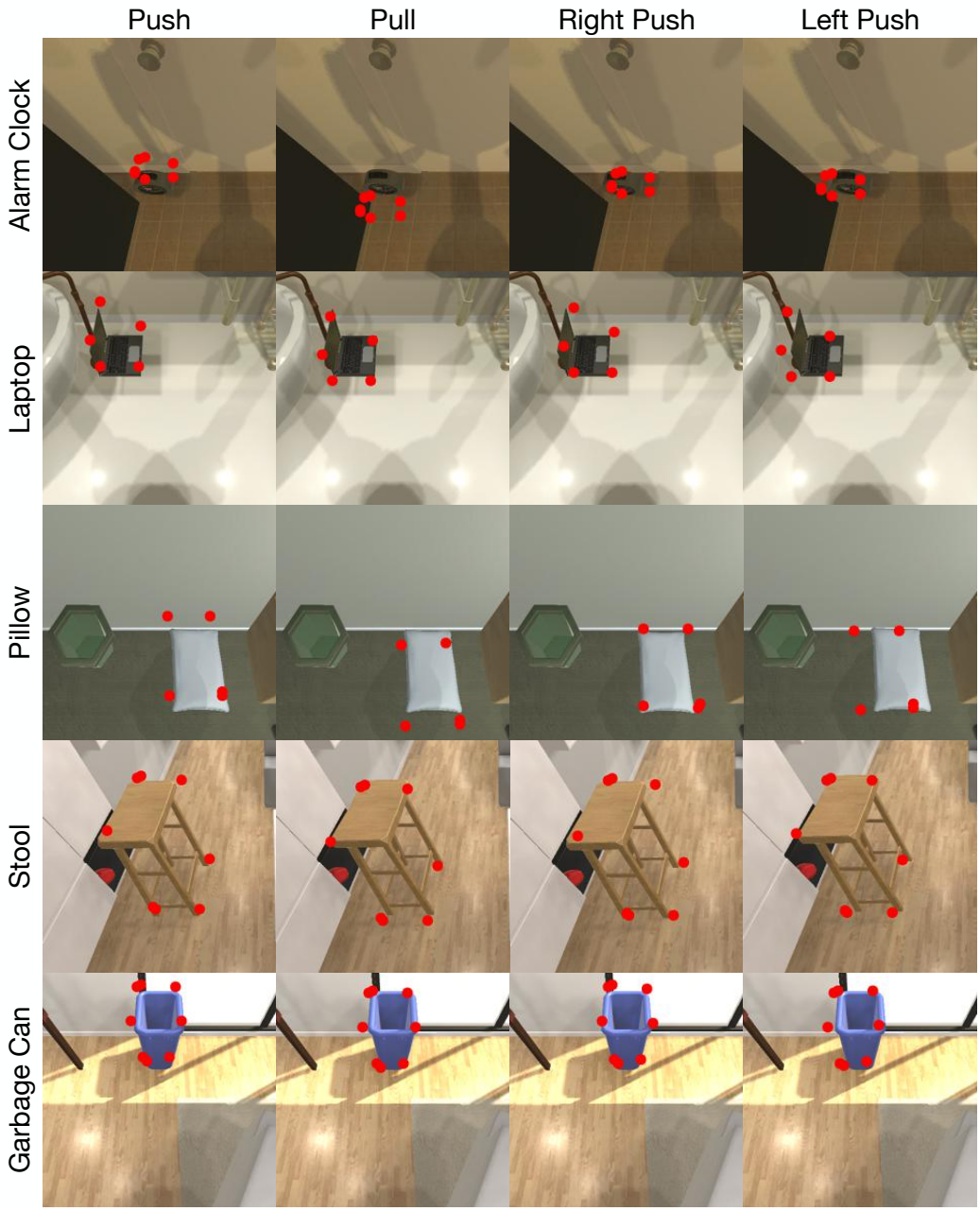

Figure 3.13: **Qualitative results of action-conditioned keypoints $\mathbf{p}^a$ prediction.** We show our action-conditioned keypoints $\mathbf{p}^a$ prediction results over 4 actions on 5 objects in 4 different testing scene (from top to bottom: `Kitchen27`, `Bathroom430`, `Bedroom328`, and `LivingRoom227`). The predicted keypoints are shown in red color.

3.10 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 pushing an object to a target point. 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.

Chapter 4

ROBUST NAVIGATION: ACTION IMPACT MODELING VIA VISUAL FORECASTING

A common assumption when training embodied agents is that the impact of taking an action is stable; for instance, executing the "move ahead" action will always move the agent forward by a fixed distance, perhaps with some small amount of actuator-induced noise. This assumption is limiting; an agent may encounter settings that dramatically alter the impact of actions: a move ahead action on a wet floor may send the agent twice as far as it expects and using the same action with a broken wheel might transform the expected translation into a rotation. Instead of relying that the impact of an action stably reflects its pre-defined semantic meaning, we propose to model the impact of actions on-the-fly using latent embeddings. By combining these latent action embeddings with a novel, transformer-based, policy head, we design an Action Adaptive Policy (AAP). We evaluate our AAP on two challenging visual navigation tasks in the AI2-THOR environment and show that our AAP is highly performant even when faced, at inference-time, with missing actions and, previously unseen, perturbed action spaces. The results show that our AAP performs consistently better across various unseen drifts, and even works well when some actions are disabled at inference. Moreover, we observe significant improvement in robustness against these actions when evaluating in real-world scenarios. The code and environment are available at prior.allenai.org/projects/action-adaptive-policy.

4.1 Introduction

Humans show a remarkable capacity for planning when faced with substantially constrained or augmented means by which they may interact with their environment. For instance, a human who begins to walk on ice will readily shorten their stride to prevent slipping. Likewise, a human will spare little mental effort in deciding to exert more force to lift their hand when it is weighed down by groceries. Even in these mundane tasks, we see

that the effect of a humans' actions can have significantly different outcomes depending on the setting: there is no predefined one-to-one mapping between actions and their impact. The same is true for embodied agents where something as simple as attempting to moving forward can result in radically different outcomes depending on the load the agent carries, the presence of surface debris, and the maintenance level of the agent's actuators (*e.g.*, are any wheels broken?). Despite this, many existing tasks designed in the embodied AI community [15, 34, 43, 54, 67, 68, 83, 97, 108, 121, 156, 183, 197, 216, 219, 243] make the simplifying assumption that, except for some minor actuator noise, the impact of taking a particular discrete action is functionally the same across trials. We call this the *action-stability assumption* (AS assumption). Artificial agents trained assuming action-stability are generally brittle, obtaining significantly worse performance, when this assumption is violated at inference time [32]; unlike humans, these agents cannot adapt their behavior without additional training.

In this chapter, we study how to design a reinforcement learning (RL) policy that allows an agent to adapt to significant changes in the impact of its actions at inference time. Unlike work in training robust policies via domain randomization, which generally leads to learning conservative strategies [111], we want our agent to fully exploit the actions it has available: philosophically, if a move ahead action now moves the agent twice as fast, our goal is not to have the agent take smaller steps to compensate but, instead, to reach the goal in half the time. While prior works have studied test time adaptation of RL agents [111, 146, 223, 237], the primary insight in this chapter is an action-centric approach which requires the agent to generate action embeddings from observations on-the-fly (i.e., no pre-defined association between actions and their effect) where these embeddings can then be used to inform future action choices.

In our approach, summarized in Fig. 4.1, an agent begins each episode with a set of unlabelled actions $\mathcal{A} = \{a^0, ..., a^n\}$. Only when the agent takes one of these unlabelled actions a^i at time t, does it observe, via its sensor readings, how that action changes the agent's state and the environment. Through the use of a recurrent action-impact encoder module, the agent then embeds the observations from just before (o_t) and just after (o_{t+1}) taking the action to produce an embedding of the action $e_{i,t}$. At a future time step t',

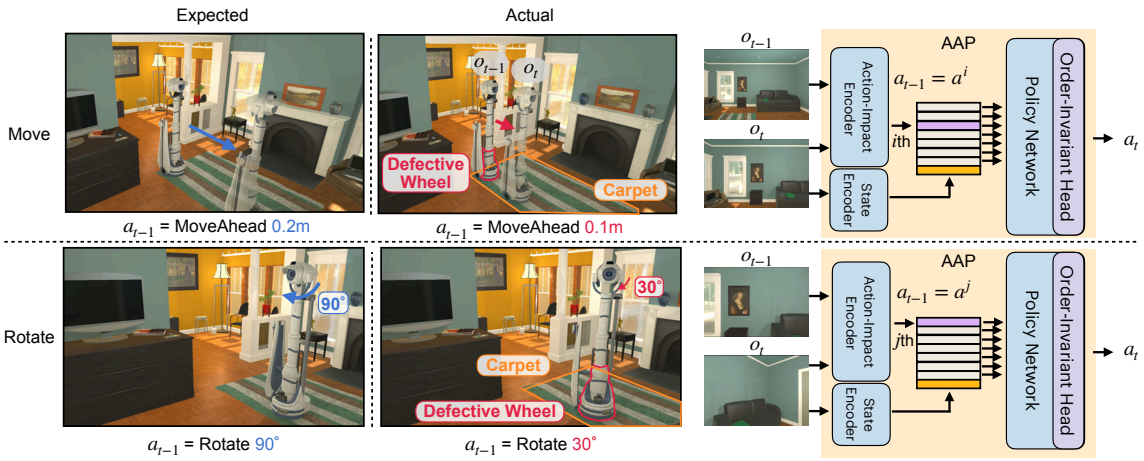

Figure 4.1: **Robust Navigation: Action Impact Modeling via Visual Forecasting.** An agent may encounter unexpected drifts during deployment due to changes in its internal state (*e.g.*, a defective wheel) or environment (*e.g.*, hardwood floor *v.s.* carpet). Our proposed Action Adaptive Policy (AAP) introduces an action-impact encoder which takes state-changes (e.g., $o_t \rightarrow o_{t+1}$) caused by agent actions (e.g., a_{t-1}) as input and produces embeddings representing these actions' impact. Using these action embeddings, the AAP utilizes a Order-Invariant (OI) head to choose the action whose impact will allow it to most readily achieve its goal.

the agent may then use these action-impact embeddings to choose which action it wishes to execute. In our initial experiments, we found that standard RL policy heads, which generally consist of linear function acting on the agent's recurrent belief vector b_t, failed to use these action embeddings to their full potential. As we describe further in Sec. 4.4.4, we conjecture that this is because matrix multiplications impose an explicit ordering on their inputs so that any linear-based actor-critic head must treat each of the $n!$ potential action orderings separately. To this end, we introduce a novel, transformer-based, policy head which we call the Order-Invariant (OI) head. As suggested by the name, this OI head is invariant to the order of its inputs and processes the agent's belief jointly with the action embeddings to allow the agent to choose the action whose impact will allow it to

most readily achieve its goal. We call this above architecture, which uses our recurrent action-impact encoder module with our OI head, the Action Adaptive Policy (AAP).

To evaluate the AAP, we train agents to complete two challenging visual navigation tasks within the AI2-THOR environment [107]: Point Navigation (PointNav) [43] and Object Navigation (ObjectNav) [43][12]. For these tasks, we train models with moderate amounts of simulated actuator noise and, during evaluation, test with a range of modest to severe unseen action impacts. These include disabling actions, changing movement magnitudes, rotation degrees, *etc.*; we call these action augmentations *drifts*. We find that, even when compared to sophisticated baselines, including meta-learning [223], a model-based approach [243], and RMA [111], our AAP approach handily outperforms competing baselines and can even succeed when faced with extreme changes in the effect of actions. Further analysis shows that our agent learns to execute desirable behavior at inference time; for instance, it quickly avoids using disabled actions more than once during an episode despite not being exposed to disabled actions during training. In addition, the experimental results (Sec. 4.12) in a real-world test scene from RoboTHOR on the Object Navigation task demonstrate that our AAP performs better than baselines against unseen drifts.

In summary, our contributions include: (1) an action-centric perspective towards test-time adaptation, (2) an Action Adaptive Policy network consisting of an action-impact encoder module and a novel order-invariant policy head, and (3) extensive experimentation showing that our proposed approach outperforms existing adaptive methods.

4.2 Related Work

4.2.1 Adaptation.

There are various approaches in the literature that address adaptation to unknown environments, action effects, or tasks.

---

[1] We also show results in a modified PettingZoo environment [199]: Point Navigation and Object Push in Sec. 4.10.

[2] We also show results in a modified Habitat environment [177]: Point Navigation in Gibson v1 in Sec. 4.11.

4.2.2 Novel environments

These approaches explore the adaptation of embodied agents to unseen environment dynamics [1, 56, 111, 112, 124, 125, 129, 155, 161, 187, 223, 236, 244, 249]. Various techniques such as meta-learning [223], domain randomization [236], and image translation [124], have been used for adaptation. In contrast to these approaches, we address changes in the actions of the agent as well. Moreover, unlike some of these approaches, *e.g.* [124], we do not assume access to the test environment.

4.2.3 Damaged body and perturbed action space

These methods focus on scenarios that the effect of the actions changes during inference time as a result of damage, weakened motors, variable load, or other factors. [237] study adaptation to a weakened motor. [146] explore adaptation to a missing leg. [235] adapt to differences in mass and inertia of the robot components. Our approach is closer to those of [237] and [146] that also consider changes in environment structures as well. Nonetheless, these works focus on using meta-learning for locomotion tasks and optimize the model at the testing phase while we do not optimize the policy at inference time. In our experiments, we find we outperform meta-learning approaches without requiring, computationally taxing, meta-learning training.

4.2.4 Novel tasks

Several works focus on adapting to novel tasks from previously learned tasks or skills [35, 58, 61, 75, 93]. We differ from these methods as we focus on a single task across training and testing.

4.2.5 Out-of-distribution generalization

Generalization to out-of-distribution test data has been studied in other domains such as computer vision [89, 160], NLP [133, 245], and vision & language [2, 5]. In this chapter, our focus is on visual navigation, which in contrast to the mentioned domains, is an interactive task and requires reasoning over a long sequence of images and actions.

4.2.6 System identification

Our idea shares similarities with the concept of System Identification [13, 19, 38, 120, 180, 202]. The major difference between our approach and the mentioned works is that we use visual perception for adaptation.

4.3 Problem Formulation

In this chapter, we aim to develop an agent which is robust to violations of the AS assumption. In particular, we wish to design an agent that quickly adapts to settings where the outcomes of actions at test time differ, perhaps significantly, from the outcomes observed at training time; for instance, a rotation action might rotate an agent twice as far as it did during training or, in more extreme settings, may be disabled entirely. We call these unexpected outcomes of actions, *drifts*. As discussed in Sec. **??**, the AS assumption is prevalent in existing embodied AI tasks and thus, to evaluate how well existing models adapt to potential drifts, we must design our own evaluation protocol. To this end, we focus on two visual navigation tasks, Point and Object Navigation (PointNav and ObjectNav), as (1) visual navigation is a well-studied problem with many performant, drift-free or fixed-drift, baselines, and (2) the parsimonious action space used in visual navigation (`Move`, `Rotate`, and `End`) allows us to more fully explore the range of drifts possible for these tasks. We will now describe the details of how we employ drifts to evaluate agents for these tasks.

In this chapter, we assume that a particular drift, perhaps caused by a defective sensor, broken parts, motor malfunction, different amount of load, or stretched cable [20], may change across episodes. We focus primarily on two categories of potential drift occurring in visual navigation tasks: movement drift d_m, which causes an unexpected translation when executing a `Move` action, and rotation drift d_r, which leads to an unexpected rotation when executing a `Rotate` action. More specifically, we semantically define the movement and rotation actions as `Move(`d`)=` "move forward by d meters" and `Rotate(`θ`)=` "rotate by θ degrees clockwise". As we describe below, the semantic meaning of these actions is designed to be true, up to small-to-moderate noise, during training, but may change significantly during evaluation.

For each training and evaluation episode, a movement drift d_m and a rotation drift d_r are sampled and fixed throughout the episode. At every step, upon executing `Move(`d`)` the agent experiences a $d + d_m + n_d$ translation toward its forward-facing direction, where the n_d represents the high-frequency actuator noise from the RoboTHOR environment [43]. Similarly, when the agent executes `Rotate(`θ`)`, the agent rotates by $\theta + d_r + n_\theta$ degrees where n_θ again represents high-frequency actuator noise. To evaluate robustness to the AS assumption, the strategies we use to choose d_m and d_r differ between training and evaluation episodes.

• **During training.** At the start of a training episode, we sample movement drift $d_m \sim U(-p, p)$ and rotation drift $d_r \sim U(-q, q)$, where $U(\cdot, \cdot)$ denotes a uniform distribution and $|p| \ll u_m$ as well as $|q| \ll 180°$. We set $u_m = 0.25$m, the standard movement magnitude in RoboTHOR.

• **During inference.** At the start of an evaluation episode, we sample movement drift $d_m \notin [-p, p]$ and a rotation drift $d_r \notin [-q, q]$.

Note that the drifts chosen during evaluation are disjoint from those seen during training.

4.4 Action Adaptive Policy (Model)

We first overview our proposed Action Adaptive Policy (AAP) in Sec. 4.4.1. We then present the traditional state encoder used for Embodied Agent policy in Sec. 4.4.2. We further introduce the details of our action-impact encoder and Order-Invariant (OI) head in Sec. 4.4.3 and Sec. 4.4.4, respectively. Finally, we describe our training strategy in Sec. 4.4.5.

4.4.1 Model Overview

The goal of our model is to adapt based on action impacts it experiences at test time. To accomplish this, the Action Adaptive Policy (AAP) includes three modules: a state encoder, an action-impact encoder, and a policy network with an order-invariant head (OI head), as illustrated in Fig. 4.2. The state encoder is responsible for encoding the agent's current observation, including a visual observation v_t and the task goal, into a hidden state h_t. The action-impact encoder processes the current observation, previous observation, and previous action a_{t-1} to produce a set of action embeddings E_t, one embedding for each action

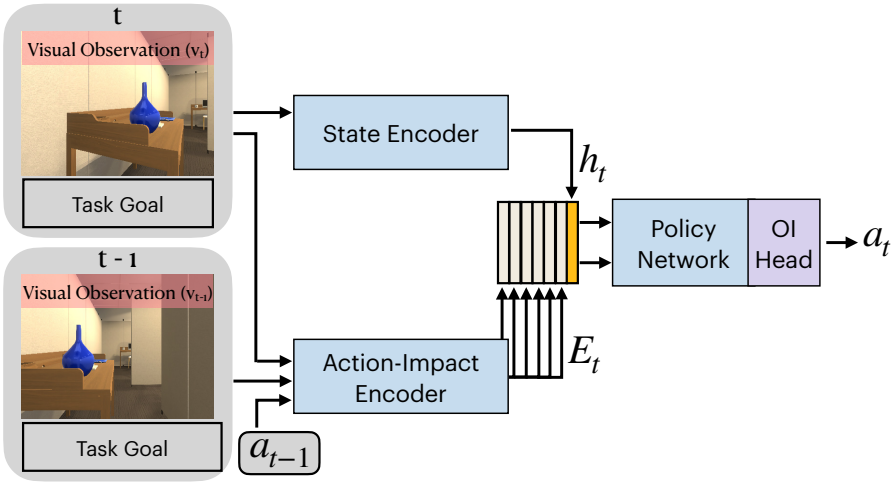

Figure 4.2: **Model Overview.** Our model includes three modules: a state encoder, an action-impact encoder, and a policy network with an order-invariant head (OI head). The blue and purple colors denote learnable modules, and the yellow and light gray color represents hidden state from the state encoder and action embedding from the action-impact encoder.

in the action space $\mathcal{A}$. More specifically, the job of the action-impact encoder is to update the action embeddings via the previous action recurrently using the encoded *state-change* feature (Sec. 4.4.3). Note that the embeddings $e_{i,t} \in E_t$ are not given information regarding which action was called to produce them; this is done precisely so that the agent cannot memorize the semantic meaning of actions during training and, instead, must explicitly model the impact of each action. Finally, the policy network with the OI head takes both the hidden state h_t and the set of action embeddings E_t as inputs and predicts an action a_t.

4.4.2 State Encoder

Following [16, 102, 220], we implement the state encoder with a visual encoder and an embedder, as shown in Fig 4.3. Note that popular baseline policy models used for embodied AI tasks frequently employ a policy network directly after the state encoder to produce the

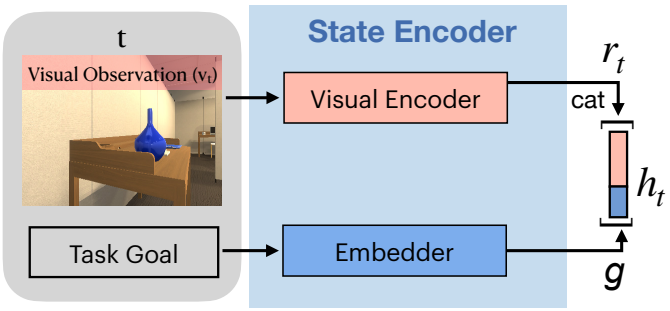

Figure 4.3: **State Encoder** is composed of a visual encoder for visual encoding and a embedder for task goal. The light red and dark blue colors indicate the learnable visual encoder and goal embedder, respectively.

agent's action. In this chapter, we use an RGB image as our visual observation v_t. For the *PointNav* task goal, we follow [177] and use both GPS and compass sensors to provide the agent with the angle and distance to the goal position, $\{\Delta\rho_t, \Delta\phi_t\}$. For the *ObjectNav* task, we follow [43] and provide the agent with a semantic token corresponding to the goal object category as the task goal. We use a CLIP-pretrained ResNet-50 [86, 167] as our visual encoder and a multi-layer perceptron (MLP) as the embedder to obtain visual representation r_t and goal representation g, respectively. Finally, we concatenate them to construct the hidden state h_t.

4.4.3 Action-Impact Encoder

In this section, we introduce the action-impact encoder, see Fig. 4.4. The goal of this encoder is to produce a set of action embeddings E_t which summarize the impact of actions the agent has taken in the episode so far. To adapt to unseen drifts during inference, these action embeddings should not overfit to action semantics, instead they should encode the impact that actions actually have at inference time. The action-impact encoder first extracts a state-change feature f from two consecutive observations. It then utilizes a recurrent neural network (RNN) to update the action embedding $e_{i,t}$ according to the previous action $a_{t-1}{=}a^i$. As the action-impact encoder generates embeddings for each action a^i without

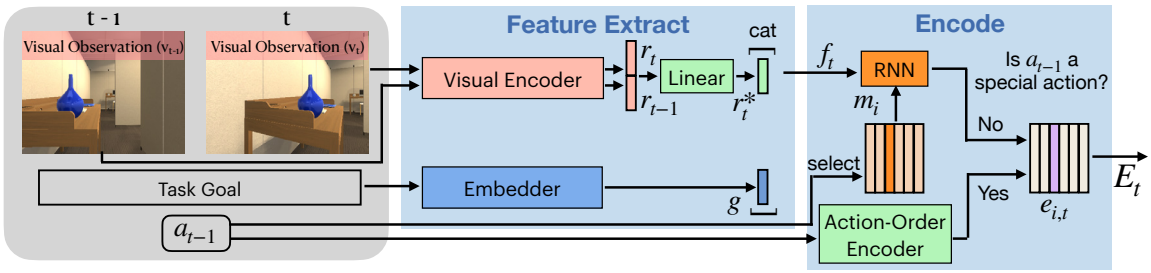

Figure 4.4: **Action-Impact Encoder.** The input to the action-impact encoder are two consecutive observations and the previous action. The encoder first extracts visual representations and a goal representation via a ResNet-50 and an Embedder. Concatenated, these form a state-change feature f_t. The encoder then uses the previous action a_{t-1}=a^i to retrieve the corresponding memory m_i. With m_i, an RNN maps f_t to an embedding. Finally, the encoder registers this embedding as the action embedding $e_{i,t}$ if a^i is not a "special" action (*i.e.*, a non-actuator-based action). Otherwise, the encoder registers an action embedding obtained from the Action-Order Encoder into $e_{i,t}$.

using semantic information about the action, every embedding $e_{i,t}$ only encodes the effect of its corresponding action. The decision to not use information about action semantics has one notable side-effect: at the start of an episode, all action embeddings $e_{i,0}$ will be equal so that the agent will not know what any of its actions accomplish. It is only by using its actions, and encoding their impact, that the agent can learn how its actions influence the environment.

During the feature extraction stage, we employ the same visual encoder and embedder used by the state encoder to process the visual observations, v_t and v_{t-1}, and the task goal. A linear layer is used to project the two concatenated visual representations $[r_t, r_{t-1}] \in \mathbb{R}^l$ to $r_t^* \in \mathbb{R}^{\frac{l}{2}}$, where $l = 1024$ for *PointNav* and $l = 3136$ for *ObjectNav*. We then concatenate r_t^* and the goal representation g, to form the state-change feature $f_t = [r_t^*, g]$, see the *Feature Extract* stage of Fig. 4.4.

After feature extraction, we apply an RNN, namely a GRU [36], to summarize state

changes through time, as illustrated in the *Encode* stage of Fig. 4.4. The use of a recurrent network here allows the agent to refine its action embeddings using many observations through time and also ignore misleading outliers (*e.g.*, without a recurrent mechansim, an agent that takes a `Move` action in front of a wall may erroneously believe that the move action does nothing). To update the action embeddings recurrently, we utilize the previous action $a_{t-1} = a^i$ to index into a matrix of memory vectors to obtain the latest memory m_i associated with a^i. This memory vector is passed into the RNN with f_t recurrently. In this way, the state-change resulting from action a^i only influences the embedding e_i. Note that we use the same RNN to summarize the state-changes for all actions. Moreover, since a_{t-1} is only used to select the memory, the RNN is agnostic to the action order and focuses only on state-change modeling. Thus, the action embedding produced by the RNN does not contain action semantics, but does carry state-changes information (*i.e.*, action impact).

One technical caveat: the PointNav and ObjectNav tasks both have a special `End` action that denotes that the agent is finished and immediately ends the episode. Unlike the other actions, it makes little sense to apply action drift to `End` as it is completely independent from the agent's actuators. We, in fact, do want our agent to overfit to the semantics of `End`. To this end we employ an Action-Order Encoder which assigns a unique action embedding to the `End` in lieu of the recurrent action embedding. Note that these types of unique action embeddings are frequently used in traditional models to encode action semantics. Finally, we register the recurrent embedding into the action embedding $e_{i,t}$ via the previous action $a_{t-1} = a^i$, if this action a^i is not the `End` (*Encode* panel of Fig. 4.4); otherwise, we register the action embedding $e_{i,t}$ as the output of the Action-Order Encoder.

4.4.4 Policy Network with an Order-invariant Head

Standard policy networks in embodied AI use an RNN to summarize state representations h_t through time and an actor-critic head to produce the agent's policy (a probability distribution over actions) and an estimate of the current state's value. Frequently, this actor-critic head is a simple linear function that maps the agent's current beliefs b_t (*e.g.*, the output of the RNN) to generate the above outputs, see Fig. 4.5 (b). As the matrix multiplications

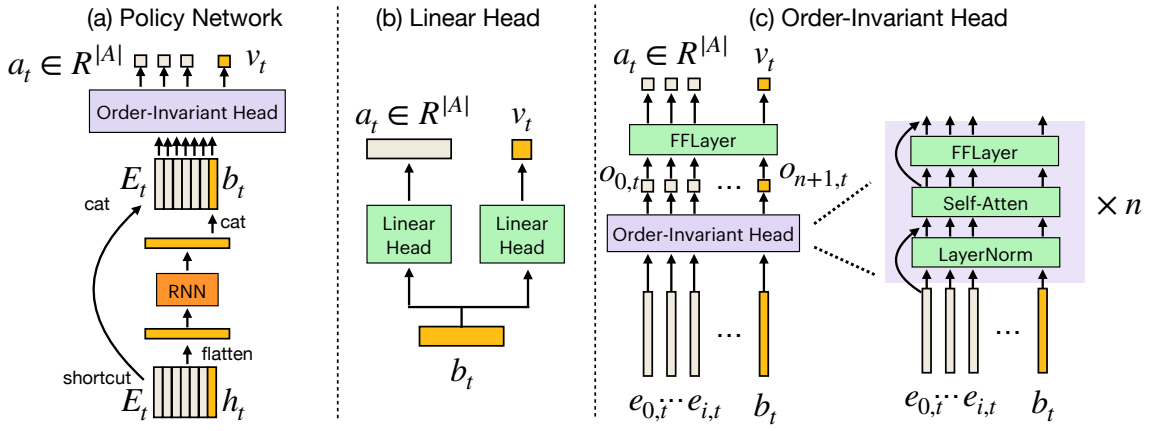

Figure 4.5: **(a) Policy Network with Order-Invariant Head** first flattens the input and uses an RNN to produce a belief b. An Order-Invariant head further processes the action embeddings E and belief b to predict action probability and value. **(b) Linear actor-critic** takes the belief b from RNN to predict action probability and value. **(c) Order-Invariant Head** is invariant to the order of its inputs so the policy predicts the action probability and value based on state-changes (*i.e.*, action impact) instead of action semantics (*i.e.*, a consistent action order).

used by this linear function require that their inputs and outputs be explicitly ordered, this imposes an explicit, consistent, ordering on the agent's actions. More specifically, let the weight matrix in the linear mapping be $W = [\mathbf{w}_0|\ldots|\mathbf{w}_n]^T$, then $\mathbf{w}_i$ are the unique set of weights corresponding to the ith action a^i. Because the $\mathbf{w}_i^T$ is specifically learned for the ith action during the training stage, it encodes the semantics of action a^i and thereby prevents the policy from generalizing to unexpected drifts applied to a^i. For example, if the action a^i is `Rotate(30°)`, and the policy is learned with training drifts in the range between $[-15°, 15°]$, then policy will fail catastrophically when experiencing an unexpected drift $20°$ because the policy has overfit to the semantics that the a^i should only be associated with $30° \pm 15°$ rotation.

To address this issue, we propose to equip our policy network with a transformer-based

Order-Invariant (OI) actor-critic head, as shown in Fig. 4.5 (a). The OI head, illustrated in Fig. 4.5 (c), takes both the belief b_t and the action embeddings $e_{0,t}, \ldots, e_{n,t}$ as input to produce $o_{0,t}, \ldots, o_{n,t}, o_{n+1,t}$, where the $o_{n+1,t}$ is produced by b_t. A Feed-Forward Layer then maps the $o_{0,t}, \ldots, o_{n,t}$ to action logits and the $o_{n+1,t}$ to an estimate of the current state's value. Finally, we apply Softmax on the action logits to generate action probabilities. We implement the OI head using a Transformer [200] encoder without a positional encoding. By design, the extensive weight sharing in a transformer (*e.g.*, the query, key, and value embedding, as well as the Feed-Forward Layer are shared across all the input tokens) means that there are no unique weights assigned to any particular action.

In conclusion, as (1) the transformer uses the same weights for each input embedding, (2) the removal of positional encoding prevents the model from focusing on action semantics, and (3) the set of action embeddings E is used to represent the impact of actions, the proposed AAP can choose the action most useful for its goal despite unexpected drifts. In Sec. 4.8 we show the importance of weight sharing and the action embeddings E by comparing our overall model with two ablations: (i) AAP but with a linear actor-critic head and, (ii) AAP but with no Action-Impact Encoder.

4.4.5 Training Strategy

To train our agent, we use DD-PPO [222] to perform policy optimization via reinforcement learning. To endow the action embeddings E with an ability to describe the state-change between two consecutive observations, we utilize an auxiliary model-based forward prediction loss to optimize the action-impact encoder. In particular, we apply a Feed-Forward Layer (a shared MLP applied to each input independently), which operates on each embedding $\{e_0, \ldots, e_n\}$ independently to predict the agent state-change from the current step t to the next step $t + 1$. As long as the RNN in the action-impact encoder has been exposed to state-changes resulting from an action, the action-impact encoder can learn to predict the precise agent state change Δs_{t+1}. Our forward prediction loss is optimized jointly alongside DD-PPO using the same collection of on-policy trajectories collected in standard DD-PPO training. Namely, if M consecutive agent steps are col-

lected during an on-policy rollout, then we apply a simple *MSE* Loss *w.r.t* the ground-truth agent state-changes $\Delta\mathbf{s}^* = \{\Delta s^*_{t-M}, \Delta s^*_{t-M+1}, ..., \Delta s^*_t\}$: $\mathcal{L}_{\text{forward}} = MSE(\Delta\mathbf{s}, \Delta\mathbf{s}^*)$, where $\Delta\mathbf{s} = \{\Delta s_{t-M}, \Delta s_{t-M+1}, ..., \Delta s_t\}$ are the predicted agent state-changes. Therefore, our overall learning objective is $\mathcal{L} = \mathcal{L}_{\text{PPO}} + \alpha\mathcal{L}_{\text{forward}}$, where α controls the relative importance of $\mathcal{L}_{\text{forward}}$. We set $\alpha = 1.0$ in this chapter. For model optimization details, see Fig. 4.6 in the Sec. 4.5.2.

4.5 Experiments

In our experiments, we aim to answer the following questions: (1) How does the proposed AAP perform when exposed to expected or modest drifts and how does this performance compare to when it is exposed to unseen or severe drifts? (2) Can the AAP handle extreme cases where some actions are disabled and have no effect on the environment? (3) When faced with multiple disabled actions, can the AAP recognize these actions and avoid using them? (4) Is it important to use the action-impact encoder and OI head jointly or can using one of these modules bring most of the benefit? (5) Qualitatively, how does the AAP adapt to, previously unseen, action drifts?

4.5.1 Implementation details.

We consider two visual navigation tasks, *PointNav* and *ObjectNav*, using the RoboTHOR framework [43]. RoboTHOR contains 75 scenes replicating counterparts in the real-world and allows for rich robotic physical simulation (*e.g.*, actuator noise and collisions). The goals of the two tasks are for the agent to navigate to a given target; in *PointNav* this target is a GPS coordinate and in *ObjectNav* this target is an object category. In the environment, we consider 16 actions in action space $\mathcal{A} = \{$`Move(d)`, `Rotate(`θ`)`, `End`$\}$, where `d` $\in \{0.05, 0.15, 0.25\}$ in meters and $\theta \in \{0°, \pm30°, \pm60°, \pm90°, \pm120°, \pm150°, 180°\}$. We set $p = 0.05$m so that we sample movement drifts $d_m \sim U(-0.05, 0.05)$ and $q = 15°$ so that we sample rotation drifts $d_r \sim U(-15°, 15°)$ during training. We then evaluate using drifts $d^*_m \in \{\pm0.05, \pm0.1, 0.2, 0.4\}$ and $d^*_r \in \{\pm15°, \pm30°, \pm45°, \pm90°, \pm135°, 180°\}$. Note that only $d_m = \pm0.05$ and $d_r = \pm15°$ are drifts seen during training. The agent is considered to have successfully completed an episode if it takes the `End` action, which

always immediately ends an episode, and its location is within in 0.2 meters of the target for *PointNav* or if the target object is visible and within 1.0 meters for *ObjectNav*. Otherwise, the episode is considered a failure. We use the AllenAct [220] framework to conduct all the experiments. During training, we employ the default reward shaping defined in AllenAct: $R_{\text{penalty}} + R_{\text{success}} + R_{\text{distance}}$, where $R_{\text{penalty}} = -0.01$, $R_{\text{success}} = 10$, and R_{distance} denotes the change of distances from target between two consecutive steps. See Sec. 4.5.2 for training and model details.

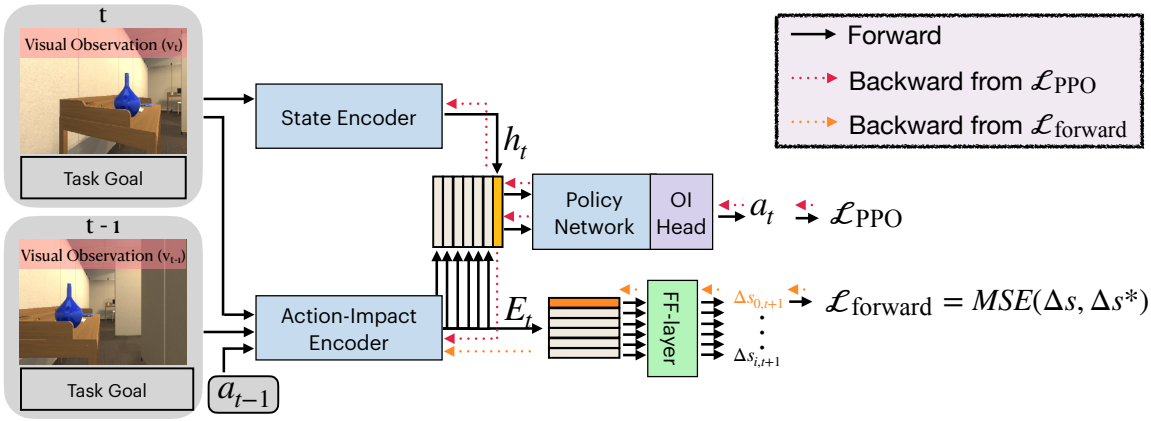

Figure 4.6: **Training Pipeline.** The forward pass is in black color, the backward from $\mathcal{L}_{\text{PPO}}$ is in red color, and the backward from $\mathcal{L}_{\text{forward}}$ is in orange color.

4.5.2 Training Pipeline, Hyperparameters, and Time Complexity

- *Training Pipeline and Hyperparameters.* The training pipeline for forward pass and backward update is shown in the Fig. 4.6. During training, we use the Adam optimizer and an initial learning rate of $3e{-}4$ that linearly decays to 0 over 75M and 300M steps for the two tasks, respectively. We set the standard RL reward discounting parameter γ to 0.99, λ_{gae} to 0.95, and number of update steps to 128 for $\mathcal{L}_{\text{PPO}}$. The α for $\mathcal{L}_{\text{forward}}$ is set to 1. Finally, we train the policy for 75M and 200M steps for *PointNav* and *ObjectNav* respectively and evaluate the model every 5 million steps.

- *Time complexity.* Theoretically, the time-complexity for the OI transformer head is $O(n^2)$, where the n is the number of actions in the action space. However, as the number of actions is usually fairly small, our AAP doe snot suffer from the same problems frequently plague transformer-based models with their n^2 complexity in the sequence length. At run-time, we evaluate models by a personal desktop with a Intel i9-9900K CPU, 64G DDR4-3200 RAM, 2 Nvidia RTX 2080 Ti GPUs. The framework (w/ our AAP) spends ≈ 35 minutes evaluating 1.8k val episodes with 5 parallel processes on the Point Navigation task. The average episode length is ≈ 117. As a result, the FPS (or interaction per second) is ≈ 100.3. During the training phase, we used an AWS machine with 48 vCPUs, $187G$ RAM, and 4 Nvidia Tesla T4 GPUs to train the policy. The FPS (or interaction per second) is ≈ 400 for our AAP.

4.6 Baselines

Each baseline method uses the same visual encoder, goal embedder, RNN and linear actor-critic in the policy network unless stated otherwise. We consider following baselines.

4.6.1 - EmbCLIP

EmbCLIP [102] is a baseline model implemented in AllenAct that uses a CLIP-pretrained ResNet-50 visual backbone. It simply uses a state-encoder to obtain hidden state h_t, applies a RNN to obtain recurrent belief b_t over h_t, and uses a linear actor-critic head to predict an action a_t via b_t. This architecture is used by the current SoTA agent for RoboTHOR ObjectNav without action drift [45].

4.6.2 - Meta-RL

Meta-RL [223] is an RL approach based on Meta-Learning [61]. However, since we do not provide reward signal during inference, we cannot simply apply an arbitrary meta-RL method developed in any prior work. Thus, we follow [223] to employ the same $\mathcal{L}_{\text{forward}}$ in our AAP as the meta-objective during training and inference. We add an MLP after the belief b_t in the EmbCLIP baseline policy to predict the next agent state-change Δs_{t+1} to

enable the meta-update phase during both training and inference stages.

4.6.3 - RMA

RMA [111] is a method assessing environmental properties using collected experiences. Although they focus on locomotion tasks, we adapt their idea into our studied problem. At the first stage in RMA training, we input both the movement drift d_m and rotation drift d_r into the model to learn a latent code $c \in \mathcal{R}^8$ of the environment. The policy network then takes c as an additional representation to make a prediction. During the second training stage, we follow [111] to learn a 1-D Conv-Net that processes the past 16 agent states to predict c. In this stage, all modules are frozen, except for the 1-D Conv-Net.

4.6.4 - Model-Based

Model-Based [243] is a model-based forward prediction approach for the agent state-changes $\Delta\mathbf{s}$. However, they focus on predicting the future agent state-changes associated with different actions, and further embed them into action embeddings. Moreover, they use a linear actor-critic to make a prediction which does not fully utilize the advantage of our action-centric approach.

4.7 Ablation Studies

We investigated which module in the AAP contributes the most to the performance, or if they are all necessary and complementary to one another. We conducted the ablation studies on *PointNav* by comparing our proposed AAP to following two baselines.

4.7.1 LAC

, LAC is a variant of AAP that uses a linear actor-critic head instead of our OI head. We compare our AAP to this baseline to investigate if the OI head is necessary to select an action from the encoded action-impact embedding.

4.7.2 Action-Semantics

, Action-Semantics is a variant of AAP that uses the OI head without the Action-Impact Encoder. We follow the traditional policy used for embodied agents (*e.g.* EmbCLIP) to implement unique learnable embeddings for each unique action. However, since this learnable embeddings are fixed after the training stage, they do not encode the state-changes on-the-fly from the latest observations at inference. This, this baseline shows the importance of the proposed action-impact embedding.

4.8 Results

4.8.1 Evaluation Metrics

We evaluate all models by their *Success Rate (SR)*; we also report the popular *SPL* [8], *Episode Length*, *Reward*, *Distance to Target*, and *soft-SPL* [40] metric in Sec. 4.9.2. *SR* is the proportion of successful episodes over the validation set. In addition, we report the *Avoid Disabled Actions Rate (ADR)* and *Disabled Action Usage (DAU)* metrics in the **Disabled Actions** experiments. The *ADR* equals the proportion of episodes in which the agent uses no disabled action more than once, and the *DAU* computes the total number of times disabled actions are used averaged across all episodes.

4.8.2 Different Drifts

The quantitative results for *PointNav* and *ObjectNav* are shown in Fig. 4.7. As shown, the *SR* of competing baselines decreases remarkably as the movement drifts d_m and rotation drifts d_r increase from modest ($d_m{=}\pm 0.05,\ d_r{=}\pm 15°$) to more severe ($d_m{=}0.4,\ d_r{=}180°$). Alternatively, our AAP performs consistently across all the drift changes and outperforms baselines significantly. Note that the baselines perform well (*SR*$\approx$1.0 for *PointNav* and *SR*$\approx$46% for *ObjectNav*) with seen drifts ($d_m{=}0.05,\ d_r{=}15°$), but our AAP achieves consistent *SR* across all drifts because the action embeddings effectively capture the state-changes associated with actions so the OI head can thereby recognize that, for instance, action $a_i =$ `Rotate`$(30°)$ becomes another action $a_j =$ `Rotate`$(120°)$ with rotation drift $d_r = 90°$ during inference. As the magnitude of movement actions are not bounded (unlike rotation actions

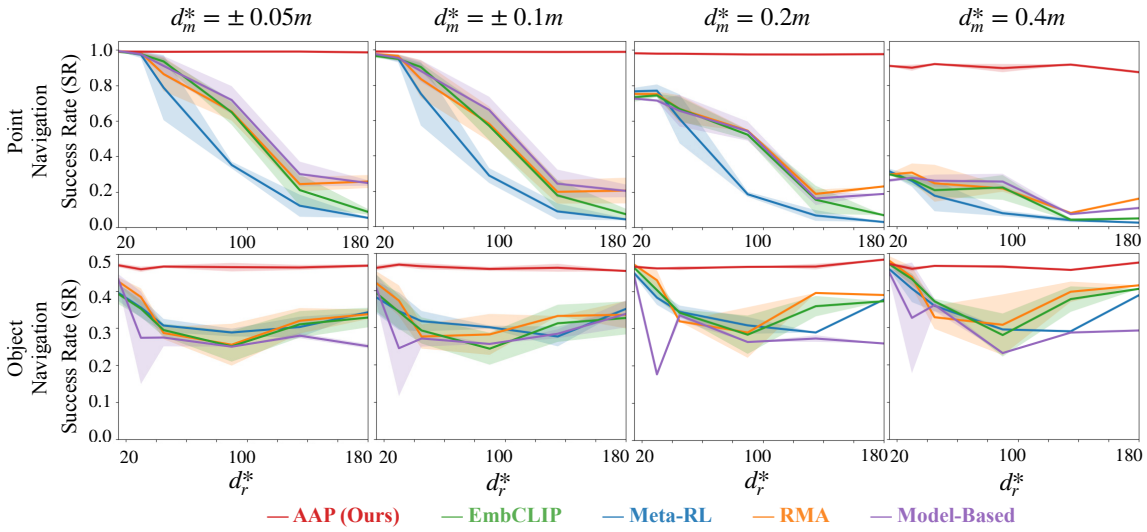

Figure 4.7: **AAP results.** Top: *PointNav* evaluation. Bottom: *ObjectNav* evaluation. We compare the proposed AAP with baselines, including EmbCLIP, Meta-RL, RMA, and Model-Based. We measure the *SR* over different drifts, including $d_m^* = \{\pm 0.05, \pm 0.1, 0.2, 0.4\}$ and $d_r^* = \{\pm 15°, \pm 30°, \pm 45°, \pm 90°, \pm 135°, 180°\}$. See Sec. 4.9.1 for an example of how our AAP learns to handle unseen drifts by using the Action-Impact Encoder and OI head.

`Rotate(`θ`)`, where $\theta \in [-180°, 180°]$), AAP's performance begins to decrease for very large movement drifts (*e.g.*, $d_m = 0.4$).

4.8.3 Disabled Actions

We consider two experimental settings to evaluate how models perform when some actions are disabled (*e.g.*, due to to a malfunctioning motors or damaged wheel). In the first setting we disable the 5 rotation angles $\theta^{\text{right}} \in \{30°, 60°, 90°, 120°, 150°\}$ so that the agent cannot rotate clockwise. In the second setting, we disable $\theta^{\text{left}} \in \{-30°, -60°, -90°, -120°, -150°\}$ so that the agent can only rotate clockwise. We otherwise set $d_m = 0.2$m and $d_r = 0°$ in this experiment. The *SR* for *PointNav* and *ObjectNav* are shown in Tab. 4.1a and Tab. 4.1b.

Table 4.1: **Disabled Actions.** Success Rate (SR) in *PointNav* (a) and *ObjectNav* (b). The baselines are EmbCLIP, Meta-RL, RMA, and Model-Based. ↑ indicate if larger numbers are preferred.

(a) *PointNav* Success Rate (*SR*)

SR↑	disable θ^{left}	disable θ^{right}
AAP (Ours)	**96.0±1.1**	**98.0±0.9**
EmbCLIP	11.8±1.4	13.0±1.3
Meta-RL	29.2±4.8	28.6±3.4
RMA	12.5±1.6	12.8±4.4
Model-Based	21.5±10.7	14.0±2.8

(b) *ObjectNav* Success Rate (*SR*)

SR↑	disable θ^{left}	disable θ^{right}
AAP (Ours)	**31.2±3.3**	**38.8±3.0**
EmbCLIP	12.8±3.2	23.8±1.7
Meta-RL	18.7±1.7	26.2±1.9
RMA	9.3±6.8	24.2±2.1
Model-Based	19.2±1.2	9.9±1.2

As shown, there is a huge difference between the baselines and AAP; for instance, for the *PointNav* task, AAP achieves near 100% *SR* while the best competing baseline (Meta-RL) achieves <35%. In addition, AAP outperforms the baselines by at least 15.2% on average in *ObjectNav*. We further report the *ADR* and *DAU* metrics for this task in Tab. 4.2a, Tab. 4.2b, Tab. 4.3a, and Tab. 4.3b; from these tables we see that AAP is very effective at recognizing what actions are disabled through the proposed action embeddings E, see *ADR*, and efficiently avoids using such actions, see *DAU*. Fig. 4.8 shows an example with θ^{left} on *PointNav* and another one with θ^{right} on *ObjectNav*. See more examples in Sec. 4.9.3 and Fig. 4.15.

4.8.4 Ablation Studies

We conducted the ablation studies on *PointNav* by comparing our proposed AAP to: (1) "*LAC*", a variant of AAP that uses a linear actor-critic head instead of our OI head and (2) "*Action-Semantics*", a variant of AAP that uses the OI head without the Action-Impact Encoder (so there are unique learnable embeddings for each unique action). The results are shown in Fig. 4.9. Although *Action-Semantics* achieves similar *SR* to AAP when $d_r \in \{15°, 30°\}$, its performance starts decreasing as the drifts become more dramatic. On the

Table 4.2: **Disabled Actions.** Avoid Disabled Actions Rate (ADR) in *PointNav* (a) and *ObjectNav* (b). The baselines are EmbCLIP, Meta-RL, RMA, and Model-Based. ↑ indicate if larger numbers are preferred.

(a) Analysis of disabled actions in *PointNav*

ADR↑	disable θ^{left}	disable θ^{right}
AAP (Ours)	**12.5±1.3**	**18.4±0.9**
EmbCLIP	0.3±0.1	1.0±0.2
Meta-RL	0.7±0.4	0.6±0.5
RMA	0.4±0.1	0.5±0.4
Model-Based	0.7±0.5	0.6±0.4

(b) Analysis of disabled actions in *ObjectNav*

ADR↑	disable θ^{left}	disable θ^{right}
AAP (Ours)	**3.2±0.3**	**5.3±3.8**
EmbCLIP	0.6±0.2	2.9±0.2
Meta-RL	2.0±0.5	2.2±0.4
RMA	0.6±0.3	2.6±0.4
Model-Based	2.5±0.4	1.5±0.8

Table 4.3: **Disabled Actions.** Disabled Action Usage (DAU) in *PointNav* (a) and *ObjectNav* (b). The baselines are EmbCLIP, Meta-RL, RMA, and Model-Based. ↓ indicate if smaller numbers are preferred.

(a) Analysis of disabled actions in *PointNav*

DAU↓	disable θ^{left}	disable θ^{right}
AAP (Ours)	**27.8±3.1**	**15.7±1.4**
EmbCLIP	93.9±0.9	93.3±0.3
Meta-RL	83.1±9.8	85.2±7.5
RMA	94.4±0.8	95.2±0.7
Model-Based	89.4±7.1	96.1±1.7

(b) Analysis of disabled actions in *ObjectNav*

DAU↓	disable θ^{left}	disable θ^{right}
AAP (Ours)	**41.4±5.3**	**28.5±8.5**
EmbCLIP	90.5±3.7	71.2±1.7
Meta-RL	80.1±6.3	84.5±8.4
RMA	95.9±1.5	73.3±4.3
Model-Based	81.4±5.7	89.8±5.4

other hand, the poor *SR* of *LAC* confirms our OI head is far better able to leverage the predicted action embeddings than a linear actor-critic.

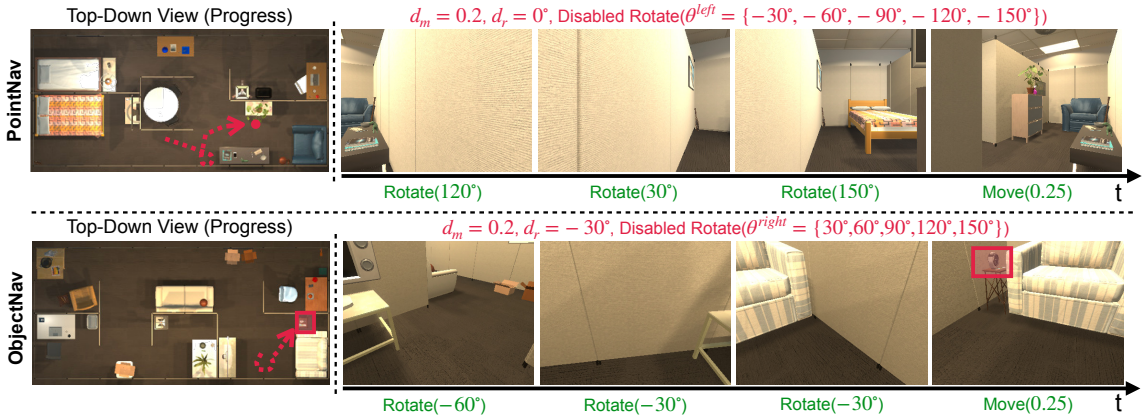

Figure 4.8: **Qualitative Results.** Examples of *PointNav* (top) and *ObjectNav* (bottom), where $d_m = 0.2$ and $d_r = 0°$ and $d_r = -30°$. Rotate left and rotate right actions are disabled, respectively. The agent adapts by rotating in the other direction to compensate for the disabled actions.

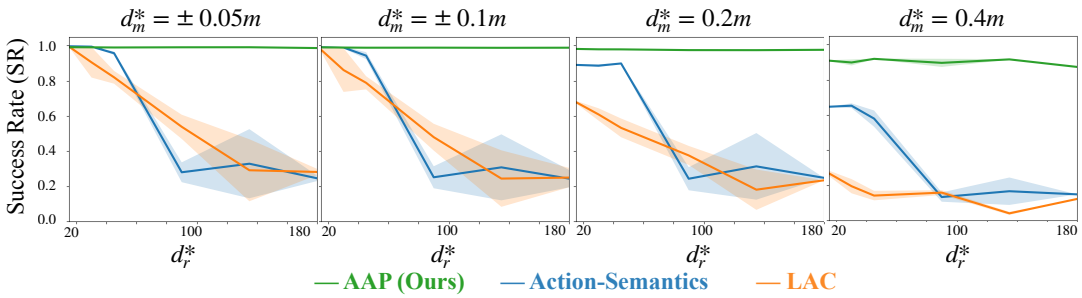

Figure 4.9: **Ablation studies.** We conducted the ablation studies on *PointNav* by comparing AAP to: (1) "*LAC*", a variant of AAP that uses a linear actor-critic head instead of our OI head and (2) "*Action-Semantics*", a variant of AAP that uses the OI head without the Action-Impact Encoder.

4.9 Analysis

4.9.1 Generalization of AAP to Unseen Drifts

We provide an example here to illustrate how our AAP generalizes to unseen drifts during the testing stage. In the training stage, the rotation actions cover all degrees $\theta \in \{-150^\circ \pm 15^\circ, -120^\circ \pm 15^\circ, ..., 150^\circ \pm 15^\circ, 180^\circ \pm 15^\circ\}$ despite with small drifts $d_r = \pm 15^\circ$, our AAP observed all possible rotation **outcomes** over $[-180^\circ, 180^\circ]$. Therefore, during the testing stage, taking $d_r^* = 180^\circ$ as an example, our policy can recognize the effect of a^i = rotate(30° + 180°drifts) is equivalent to the effect of another action a^j: rotate(-150°) which was seen in the training stage. Likewise, since the movement magnitude $d \in \{0.05 \pm 0.05, 0.15 \pm 0.05, 0.25 \pm 0.05\}$ can cover $[0, 0.3]$, our model performs well when the movement magnitude is within this range. This example highlights the core technical contributions, namely the action-impact embedding and OI head, in this chapter. With the action-impact embedding and OI head, we prevent the policy from **remembering** the action semantics, and allow it to focus on modeling the effect of actions.

4.9.2 More Quantitative Results

We show more quantitative results for *PointNav* and *ObjectNav* in Fig. 4.10, Fig. 4.11, Fig. 4.12, Fig. 4.13, and Fig 4.14 by *SPL*, *Episode Length*, *Reward*, *Distance to Target*, and *soft-SPL* As shown, the *SPL*, *soft-SPL*, and *Episode Length* of competing baselines achieve better results than AAP at the modest drifts ($d_r = \pm 15^\circ, \pm 30^\circ$), but they perform significantly worse at more severe drifts. Note that our AAP initially has no idea about the state-changes resulting from each action, it has to spend more time exploring the effects of actions in the beginning of a new episode. As a result, it results in a longer episode length comparing to the baselines with seen drifts. On the other hand, our AAP achieves consistent *SPL*, *Episode Length*, *Reward*, *Distance to Target*, and *soft-SPL* across all the drift changes and outperforms baselines remarkly. It is because AAP does not memorize action semantics, but relies on its experiences in the inference time to embed the action embeddings E on-the-fly. To collect useful experiences, the agent has to explore the environment and update the embeddings accordingly. Thereby, our AAP takes more time to understand the state-

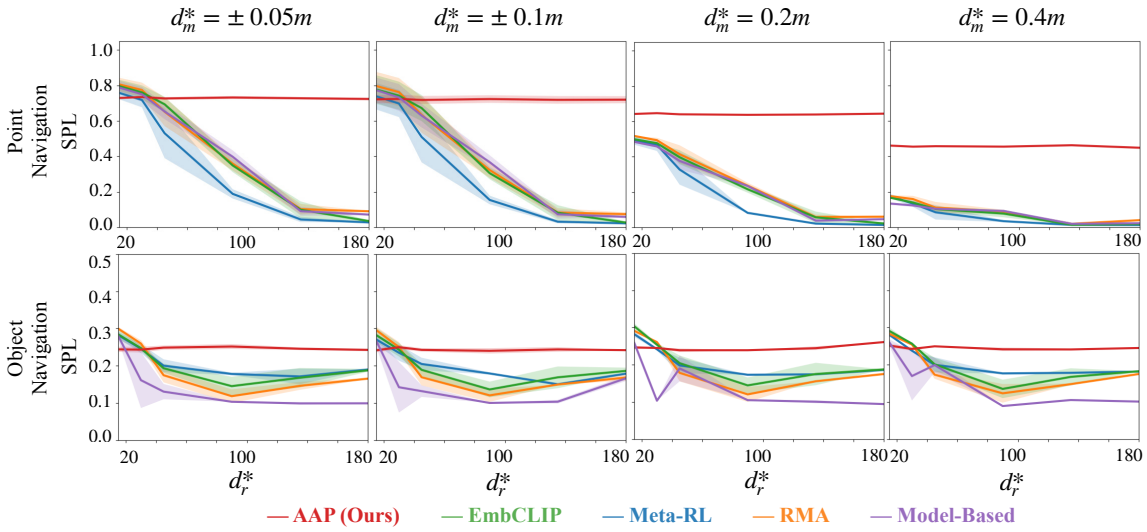

Figure 4.10: **Quantitative result using the SPL metric.** The *SPL* [8] is defined as $\frac{1}{N}\sum_{n=1}^{N} S_n \frac{L_n}{max(P_n, L_n)}$, where N is the number of episodes, S_n denotes a binary indicator of success in the episode n, P_n is the path length, and L_n is the shortest path distance in episode n. Top: *PointNav* evaluation. Bottom: *ObjectNav* evaluation.

changes, while the baselines can achieve better results based on its learned action semantics in the scenarios with the modest drifts.

- *Comparison to (Adapted) Model-Based Meta-RL [146].* Unfortunately, a direct comparison to [146] is not feasible as, (a) we focus on point navigation and object navigation in a clustered scene, but [146] focuses on locomotion control for a simple straight or curve line movement, (b) our main observation is visual observation, but [146] uses agent's state, and (c) we don't use model-predictive controller to rollout H time horizon future to make a decision, because the agent cannot access rewards or distance to the goal during the testing time, so it is not straightforward to implement a simple objective function in MPC for point navigation or object navigation task (we would argue that doing so is a research topic in itself).

However, to have a closer implementation based on the idea proposed in [146], we

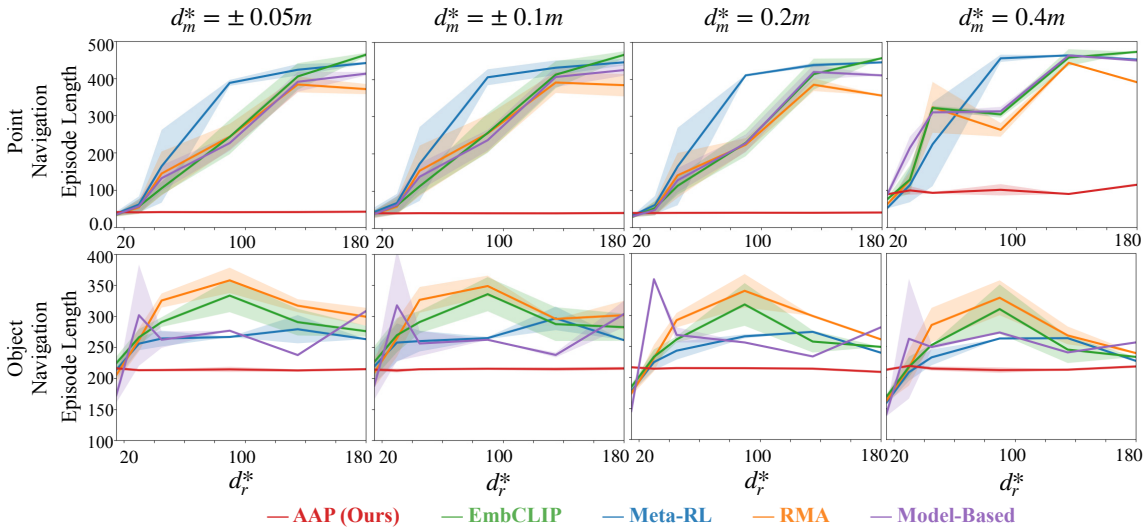

Figure 4.11: **Quantitative result using the Episode Length metric.** Top: *PointNav* evaluation. Bottom: *ObjectNav* evaluation.

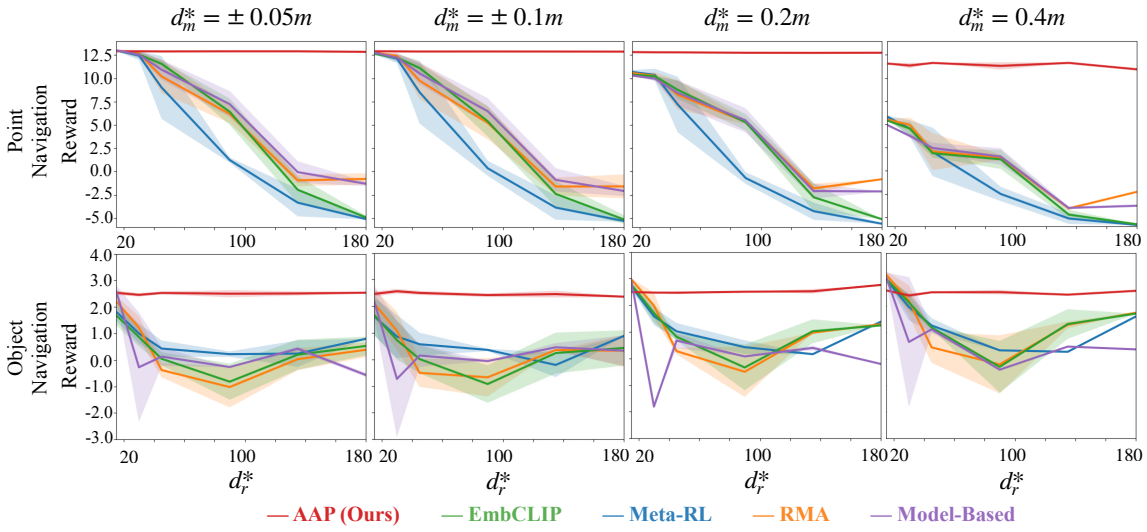

Figure 4.12: **Quantitative result using the Reward metric.** Top: *PointNav* evaluation. Bottom: *ObjectNav* evaluation.

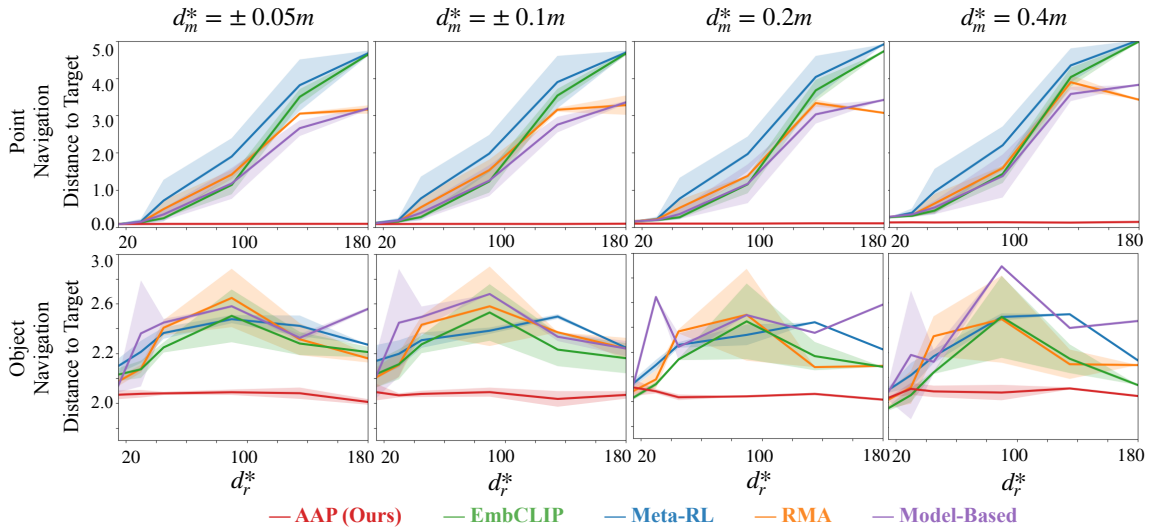

Figure 4.13: **Quantitative result using the Distance to Target metric.** Top: *PointNav* evaluation. Bottom: *ObjectNav* evaluation.

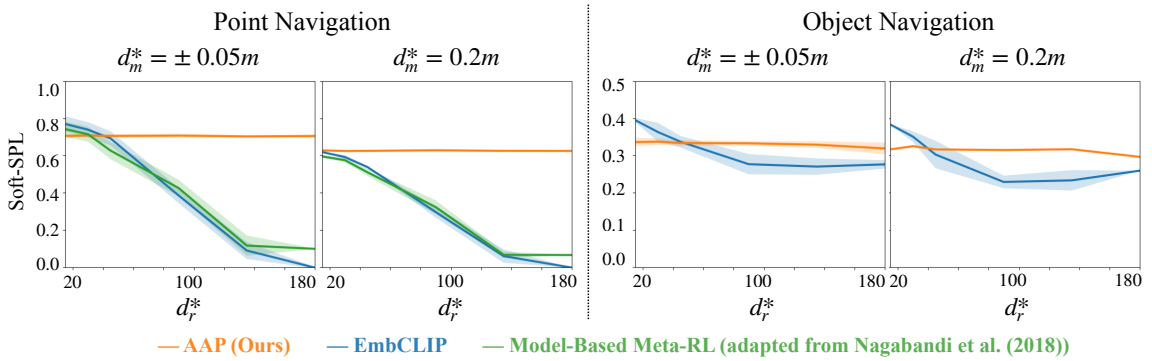

Figure 4.14: **Quantitative result using the soft-SPL.** We follow [40] to evaluate models by *soft-SPL*. *soft-SPL* is defined as $\frac{1}{N}\sum_{n=1}^{N}(1 - \frac{d_{n,\text{termination}}}{d_{n,\text{start}}})\frac{L_n}{max(P_n,L_n)}$, where N is the number of episodes, $d_{n,\text{termination}}$ and $d_{n,\text{start}}$ denote the (geodesic) distances to target upon termination and start in the episode n, P_n is the path length, and L_n is the shortest path distance in episode n. Left: *PointNav* evaluation. Right: *ObjectNav* evaluation.

combine our model-based baseline with the meta-RL baseline by employing the agent state prediction to perform meta-learning on the model-based module. The results for point navigation are shown in Fig. 4.14 in the green color by *soft-SPL*. Although this (Adapted) Model-Based Meta-RL outperforms EmbCLIP slightly facing larger drifts, it is still not able to overcome the severe drifts. It, again, highlights the effectiveness of our AAP with two major technical contributions, the action-impact embedding and the OI head, proposed in this chapter.

4.9.3 Qualitative Results

We show more qualitative results in *PointNav* and *ObjectNav* in Fig. 4.15. The first example shows the agent adopts a smaller movement magnitude to avoid the collision. The second example shows smooth moves by a right-turn, a move, another left-turn, and a final move to dodge the white table. The third example shows the agent uses three left turns with different angles to make a disabled right turn. The fourth example shows the agent slows its movement magnitude as it approaches the target object in the clustered area. The fifth example shows the agent uses two left-turn with different angles to make a disabled right turn to find the target object. The final example shows the agent uses a larger movement magnitude to move toward the target object in a relatively open area.

4.10 Results on the Modified PettingZoo Environment

We modify the PettingZoo [199] to verify the effectiveness of our AAP in a different environment with state observations only. We modify the MPE environment to simulate *PointNav* and *ObjectPush* tasks, shown in Fig. 4.16. The MPE environment provides simple simulation of collision force when objects are too close to each other. The goal for *PointNav* is to move the agent to the target location by applying acceleration to the agent. The goal for *ObjectPush* is to move the agent to push the ball to the target location by applying acceleration to the agent. The agent has to collide to the ball to perform the *push*. As shown in the figure, there are 12 actions in action space, including `Accelerate (mag`$_x$`, mag`$_y$`)`, where (`mag`$_x$`, mag`$_y$)= $\{[\pm 0.2, 0], [\pm 0.5, 0], [\pm 0.8, 0], [0, \pm 0.2], [0, \pm 0.5], [0, \pm 0.8]\}$, corresponding to 3 different accelerations towards 4 different directions. For *PointNav*, the state space is

Figure 4.15: **Qualitative Results.** Examples of *PointNav* (top tree rows) and *ObjectNav* (bottom three rows). The agent adapts by rotating in the other direction to compensate for the disabled actions.

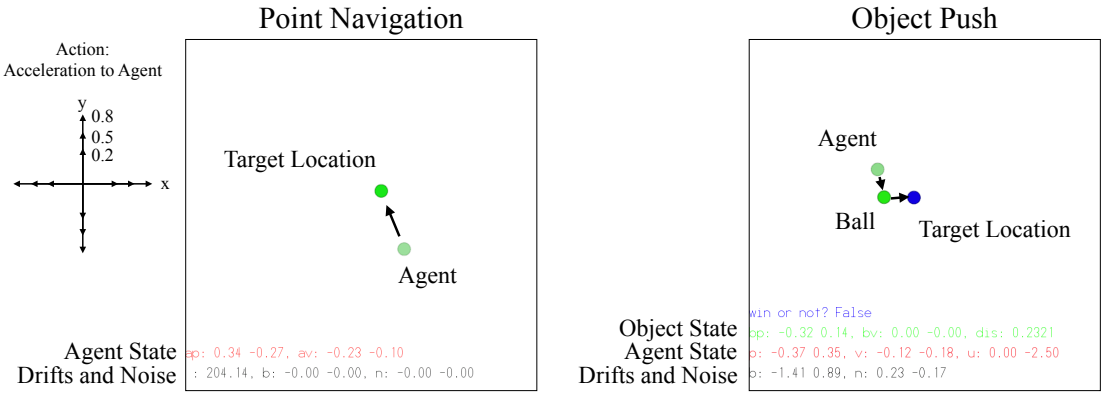

Figure 4.16: **MPE environment in PettingZoo**. We modify the MPE environment to simulate *PointNav* and *ObjectPush* tasks. The goal for *PointNav* is to move the agent to the target location and the goal for *ObjectPush* is to move the agent to push the ball to the target location. The action space consists of 3 different accelerations towards 4 directions.

6-dim, including agent's position (p_x, p_y), agent's velocity v_x, v_y, GPS sensor $(\Delta p_x, \Delta p_y)$; for *ObjectPush*, the state space is 10-dim, including agent's position (p_x, p_y), agent's velocity (v_x, v_y), object's position (o_x, o_y), object's velocity (o_{v_x}, o_{v_y}), GPS sensor $(\Delta p_x, \Delta p_y)$. Our drifts setting for training and inference stage are formulized as follows,

• **During training.** At the start of a training episode, we sample the rotation drift $d_r \sim U(-90°, 90°)$, where $U(\cdot, \cdot)$ denotes a uniform distribution.

• **During inference.** At the start of an evaluation episode, we evaluate our AAP and baselines with an unseen rotation drift $d_r^* \in \{\pm 120°, \pm 135°, \pm 150°, \pm 180°\}$.

With a rotation drift d_r, the environment applies the d_r to the input action by rotating the acceleration direction:

$$\texttt{mag}_{\text{env}} = \begin{bmatrix} cos(d_r) & sin(d_r) \\ -sin(d_r) & cos(d_r) \end{bmatrix} \texttt{mag}_{\text{input}}^T. \tag{4.1}$$

In this case, the actual direction of acceleration would be drifted according to the rotation drift d_r.

4.10.1 Models

The details about our AAP and the considered baselines are as follows,

- *GRU* simply uses a 4-layer MLP ([|state| $\rightarrow$ 64 $\rightarrow$ 64 $\rightarrow$ 64]) as the state-encoder to obtain a 64-dim hidden state h_t, applies a RNN (GRU) to obtain a 64-dim belief b_t over h_t, and uses a linear actor-critic head to predict an action a_t via b_t.

- *LAC* is a variant of our AAP that uses a linear actor-critic head instead of our OI head.

- *Model-based* is a model-based forward prediction approach for the agent state-changes $\mathbf{\Delta s}$ associated with different actions. It further embeds the prediction into action embeddings. Finally, this baseline uses a linear actor-critic to make a prediction.

- *Our AAP* uses the same 4-layer MLP used in *GRU baseline* to produce state encoding for previous state and the current state, respectively. Then, same as our AAP used in AI2THOR, the action-impact encoder uses a RNN (GRU) to produces the $n \times 64$-dim action impacts embedding E associated with different actions. Later, the model uses a RNN (GRU) in the policy obtain a 64-dim belief b_t. Finally the OI head operates on the concatenation of E and b_t to make a prediction.

4.10.2 Results

We train every model by PPO with the Adam optimizer and an initial learning rate of $1e{-}3$ that linearly decays to 0 over 20M. We set the standard RL reward discounting parameter γ to 0.99, λ_{gae} to 0.95, and number of update steps to 200 for $\mathcal{L}_{\text{PPO}}$. The α for $\mathcal{L}_{\text{forward}}$ is set to 1. Finally, we train each model by 3 different random seeds and obtain the average success rate and $1\times$ standard deviation. We evaluate models every 1M steps (checkpoint). The evaluation results are presented in Fig. 4.17. As shown in the figure, our AAP performs consistently well across all rotation drifts on both tasks in the modified MPE environment. The best baselines cannot even achieve any success rate when facing the extremest rotation drift $d_r^* = 180°$. It verifies that the effectiveness of the proposed AAP on a general reinforcement learning environment. We will release the code for this modified environment and experiments as well.

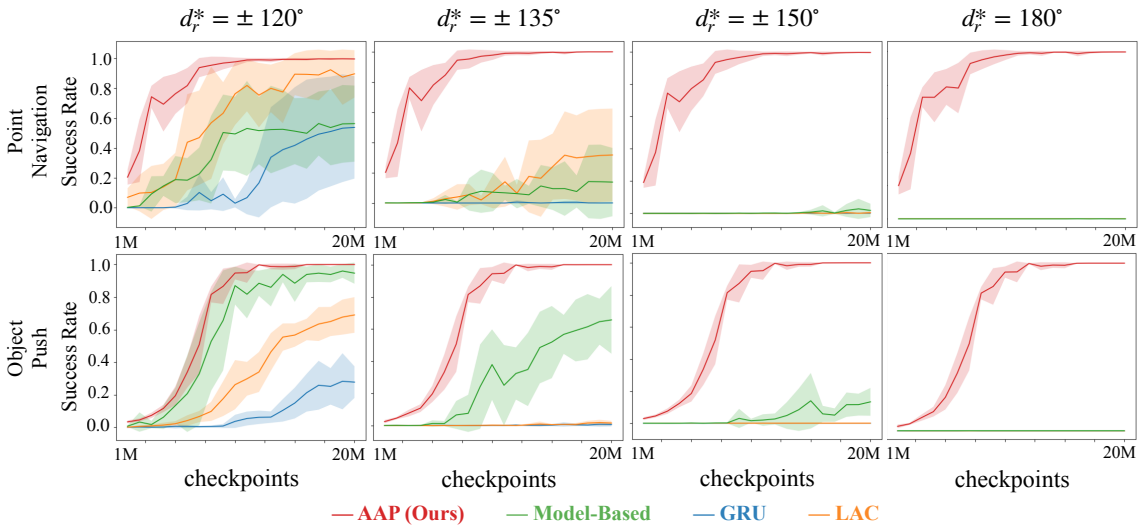

Figure 4.17: **Success Rate in MPE environment**. Top: *PointNav* evaluation. Bottom: *ObjectPush* evaluation. We train each model by 3 different random seeds and show the average success rate and 1× standard deviation in this figure.

4.11 Results on the Habitat Environment

To conduct the experiments in Habitat [177], we made a small change in the Habitat's `Move_Forward` action, where every `Move_Forward` only moves the agent by 0.01m. If the action a^i = `Move_Forward(0.25)`, the environment would move the agent by 25 times. In this way, we can implement the movement drifts as we did in AI2-THOR. For the rotation action, we do not use the default `TURN_RIGHT` or `TURN_LEFT` action. We instead directly compute the quaternion to implement the continuous rotation with rotation drifts. In this modified Habitat, we train the EmbCLIP and our AAP on Point Navigation task in Gibson `v1` [228]. The simulator is Habitat-Lab `v0.2.1`. The training settings are the same as the settings used in AI2-THOR, including the same training drifts, same learning schedule, same optimization algorithm, and learning objective. During the evaluation, we evaluate the model on movement drifts $d_m^* \in \{\pm 0.05m, \pm 0.1m, 0.2m, 0.4m\}$ and rotation drifts $d_r^* = \{\pm 15^\circ, \pm 30^\circ, \pm 45^\circ, \pm 90^\circ\}$. As shown in Fig. 4.18, our AAP performs consistently

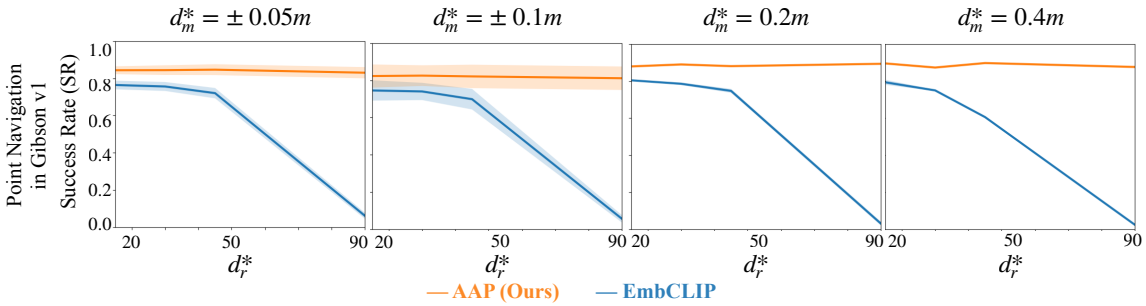

Figure 4.18: **Success Rate on Point Navigation in Habitat Environment.** We compare the proposed AAP with EmbCLIP on Point Navigation in Habitat Environment. The evaluation drifts are unseen during the training stage, including $d_m^* \in \{\pm 0.05m, \pm 0.1m, 0.2m, 0.4m\}$ and $d_r^* \in \{\pm 15°, \pm 30°, \pm 45°, \pm 90°\}$. The experiments are conducted in Habitat-Lab `v0.2.1` on Point Navigation with Gibson v1

across all movement and rotation drifts. However, the EmbCLIP is struggling with larger drifts.

4.12 Real-World Experiments.

We train our AAP on ProcTHOR 10k [45] for *ObjectNav* with training drifts d_m and d_r, then evaluate in a real-world scene from RoboTHOR [43], shown in Fig. 4.19. We include two more actions {`LookUp`, `LookDown`} into the action space to perform the horizontal angle change (30° by each execution) of the camera (along the pitch axis). For the baseline models, we compare our AAP against the EmbCLIP [102] trained on ProcTHOR 10k with the training drifts and the EmbCLIP checkpoint from the ProcTHOR project. Note that the EmbCLIP checkpoint from the ProcTHOR project was not learned with the drifts formulation during the training stage, and it only uses limited action space {`Move(0.25m)`, `Rotate(30°)`, `Rotate(−30°)`, `LookUp`, `LookDown`, `END`}. We use LoCoBot from PyRobot [145] with an LQR feedback controller [11] as our embodied agent in this real-world scene to conduct the experiments, shown in Fig. 4.20.

We then conduct an exploratory experiment with the above setting to find `Apple` and

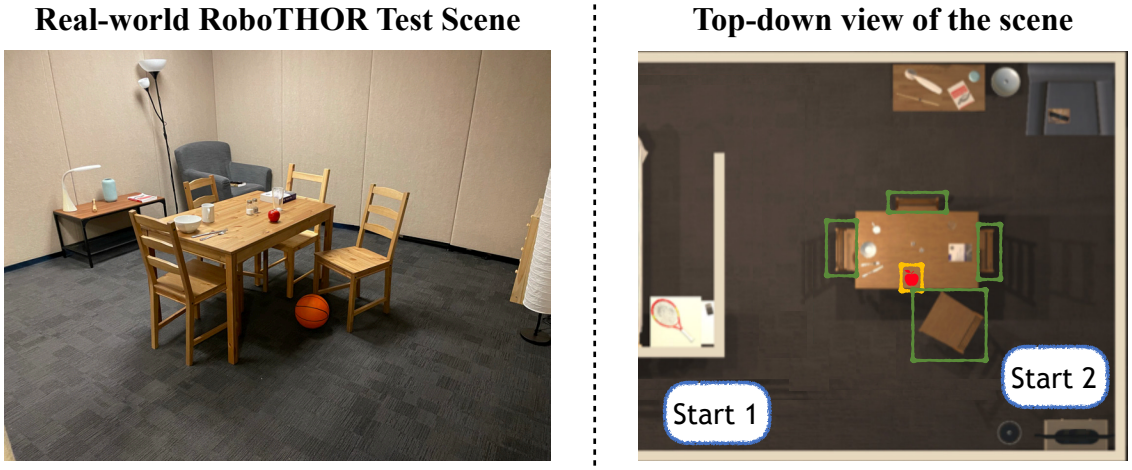

Figure 4.19: **Real-world RoboTHOR Test Scene.** Left: We show a photo of the real-world RoboTHOR Test Scene [43] used to conduct our real-world experiments. Right: It is a top-down view of the scene for visualization. For example, in this map, the target object `Apple` is marked in the yellow box, and the `Chair` is marked in the green box, while two different starting locations are marked in blue boxes.

Figure 4.20: **LoCoBot.** We use LoCoBot from PyRobot [145] with an LQR feedback controller [11] as our embodied agent in the real-world experiments.

`Chair` from 2 different starting locations within 150 steps. At inference, we inject five different drifts pairs, including $\{(d_m^* = 0m, d_r^* = 0°)$, $(d_m^* = 0.1m, d_r^* = 60°)$, $(d_m^* = 0.1m, d_r^* = -60°)$, $(d_m^* = -0.05m, d_r^* = 60°)$,$(d_m^* = -0.05m, d_r^* = -60°)\}$, by adding the drifts to the low-level commands used by the robot. Therefore, the total number of episodes used to evaluate each model is 20. Tab. 4.4 shows that AAP performs more robustly against these unseen drifts. Although the experiments show that our AAP achieves better performance than baselines, we argue that a more comprehensive study, including more types of target objects, more drifts variances, more starting locations, or with more allowed number of steps in the real-world experiments, is necessary to reach a more solid conclusion.

Table 4.4: **Real-World Experiments** results in RoboTHOR on *ObjectNav*. We compare our AAP against the EmbCLIP [102] trained on ProcTHOR 10k with the training drifts and the EmbCLIP checkpoint from the ProcTHOR project.

Model	SR↑
AAP (Ours)	65%
EmbCLIP trained *w.* drifts	30%
EmbCLIP	25%

4.13 Conclusion

We propose Action Adaptive Policy (AAP) to adapt to unseen effects of actions during inference. Our central insight is to embed the impact of the actions on-the-fly rather than relying on their training-time semantics. Our experiments show that AAP is quite effective at adapting to unseen action outcomes at test time, outperforming a set of strong baselines by a significant margin, and can even maintain its performance when some of the actions are completely disabled.

Chapter 5

CONCLUSION

In this thesis, we study how to bridge visual forecasting and model-free policy learning for interactive embodiment tasks only with visual observations. We aim to develop an embodied agent to solve a target task more efficiently, effectively, and robustly. The key idea is to forecast the future via the visual dynamics model, then incorporate the information about future rollouts to modulate the planning process in the model-free policy network. We primarily demonstrate our experimental results in physics-enabled and visually rich AI2THOR [106], Habitat [197], and RoboTHOR [43] environments[1].

In Chap. 2, we present an approach to embedding visual forecasting into a model-predictive controller with a learnable action sampler. The studied problem is *visual reaction*: the agent interacts with dynamic environments where the agent's action does not necessarily cause environmental changes. To study the problem, we formulate a task by playing catch with a drone agent, where the goal is to catch a thrown object using only visual observations. We show that using straightforward motion equations with the estimated agent's state and the predicted current object's state, we can perform accurate visual forecasting about the object's trajectory shortly. Then, the model-predictive controller can sample an action conditioning on the forecasting trajectory. Moreover, the learnable action sampler improves the overall success rate and sample efficiency (see Tab. 2.1). Finally, our proposed solution outperforms various baselines and ablations of the model, including the variations that do not use forecasting or do not learn a policy based on the forecasting. To summarize, we showcase that visual forecasting enables efficient planning so the agent can promptly accomplish a task in a changing environment.

In Chap. 3, we explore the direction of action-dependent visual forecasting, where the goal is to predict the interactive object's state-changes resulting from the agent's actions.

[1]Other than that, we also show some results in a modified PettingZoo environment [199]

The studied problem is interactive navigation, where the agent learns to change the environment to navigate more effectively to their goals. Specifically, we introduce the Neural Interaction Engine (NIE) to explicitly predict the environmental change caused by the agent's actions. We collect two datasets associated with two downstream tasks: Obstacle Navigation (ObsNav) and Object Placement (ObjPlace). The goal of the ObjNav is to reach a target location in an environment while the paths to the target are blocked. The ObjPlace requires pushing an object to a target location. The experimental results show that the NIE model achieves significant improvements over the methods without the capability of predicting the effect of actions on the surrounding environment in both scenarios (see Tab. 3.1 and Tan. 3.2). This chapter illustrates that, with visual forecasting, the agent not only learns to interact with environments but produces a plan to solve a given task effectively.

In Chap. 4, we study the problem of unexpected motion drifts at inference in the context of visual navigation. In the tasks, an agent may encounter unexpected conditions that dramatically alter the impact of actions: a move ahead action on a wet floor may send the agent twice as far as it expects. Using the same action with a broken wheel might transform the expected translation into a rotation. We propose to learn an action-impact encoder to generate an embedding from observations on-the-fly. This embedding encodes the state-changes only between the consecutive observations but does not contain the association between the actions and their effect. With the embedding, we then introduce the Order-Invariant (OI) head, constructed by a transformer model, into our policy network. Because the OI head is invariant to the order of its inputs, the policy network can process the agent's belief jointly with the embedding to choose the action whose impact will most readily achieve the agent's goal. With these designs, our proposed Action Adaptive Policy (AAP) can adapt to unseen effects of actions during inference. The final experiments show that AAP is quite robust to unseen action outcomes at test time, outperforming a set of solid baselines by a significant margin (see Fig. 4.7), and can even maintain its performance when some of the actions are entirely disabled (see Tab. 4.1a and Tab. 4.1b).

To this end, we have been considering the three design objectives to bridge visual forecasting and model-free policy learning. Nonetheless, one may ask, can we generalize a policy

for the embodied agent to unseen scenes with novel objects and out-of-distribution tasks? For example, can the agent carefully manipulate fragile objects never seen before? Besides, can we deploy the agent to the real world and solve daily tasks robustly? In short, can we learn the policy with the advantages of utilizing visual forecasting to generalize to novel and unseen scenarios?

5.1 Future Directions

This section discusses three possible future directions along the generalization topic: scaling up embodiment environments, sim-to-real deployment, and policy learning with language instructions.

5.1.1 Scale Up Embodiment Environments

The first future direction could be expanding the current scope of embodiment environments in ways such as more real-world setups [43], interaction scenarios [54], more scenes [45], and more variations of tasks [122]. We can introduce more general tasks and define detailed sub-goals and skills by presenting diverse scenes and interactions. Moreover, we can have enough scenarios where part of them needs specific requirements. For instance, when the target task is to find the key to a human when s/he is about to go out, the robot has to efficiently search the space where the key might locate and deliver it as quickly as possible. On the contrary, when cleaning up the messy counter in a kitchen, the agent may take their time and find the most appropriate cabinet to place the objects on the counter. The focus in this task becomes effective. For another example, if the assignment to the agent is to do dishes with a dishwasher, they have to be careful about the fragile plates and bowls. In this case, robustness is essential. However, without diverse scenes, observations, and tasks, we have little chance to study daily tasks that require long-horizon planning and generalization ability. Therefore, I believe a reasonable next step would be expanding the embodiment environments to another scale level.

5.1.2 Sim-to-real Deployment

The second direction is to deploy the agent from simulation environments to the real world. In the real-world scenario, the need for efficiency, generalization, and robustness amplifies. For example, no user would expect running a real-world cleaning robot at home is costly and time-consuming. Moreover, the generalization issue due to domain gaps would make the deployment more troublesome, not to mention the dangerous situations the robot may cause because the policy is not robust enough. To overcome these challenges, we could try to alleviate the domain gaps [44, 92], construct policy in modulation style [70], provide prior knowledge [48, 212, 233], or large-scale pretraining [43, 102, 182]. Among the ideas, performing image-to-image translation via the generative model [101, 173, 234] from simulation style to real-world fashion on-the-fly might be an avenue to explore. It might close the domain gaps and provide a new technique for learning good representations of motion and interactions via generative modeling. In this regard, I believe the sim-to-real deployment would be a necessary next step to verify the functionality of the learned policy with visual forecasting. In addition, it might pave the way for even newer research areas.

5.1.3 Policy Learning with Language Instructions

Thanks to recent progress in large-scale language modeling (LLM) [21, 46, 168, 169, 154], incorporating natural language processing into embodiment control has become essential and popular. Although we have already had Alfred [183], R2R [9], and RxR [109], we still need a larger-scale training scheme to learn a policy with solid generalization ability. Real-world deployment of the language-conditioned policy [4, 94] could be an interesting direction to explore along this line. Other than that, learning a policy with visual dynamics model through large-scale multimodality similar to Unified-IO [130] and Multimodal-CoT [246] could be another promising direction to enhance the generalization ability of the policy. In conclusion, I believe that learning with LLM or multimodality would be an exciting research regime to improve the robustness of the embodied agent, including its planning ability and forecasting capacity.

BIBLIOGRAPHY

[1] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *CoRL*, 2022.

[2] Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. Don't just assume; look and answer: Overcoming priors for visual question answering. In *CVPR*, 2018.

[3] 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.

[4] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *CoRL*, 2022.

[5] Arjun Reddy Akula, Soravit Changpinyo, Boqing Gong, Piyush Kumar Sharma, Song-Chun Zhu, and Radu Soricut. Crossvqa: Scalably generating benchmarks for systematically testing vqa generalization. In *EMNLP*, 2021.

[6] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, 2016.

[7] 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.

[8] 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.

[9] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018.

[10] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NeurIPS*, 2017.

[11] Karl J Åström. In *Introduction to stochastic control theory*, 2012.

[12] Lamberto Ballan, Francesco Castaldo, Alexandre Alahi, Francesco A. N. Palmieri, and Silvio Savarese. Knowledge transfer for scene-specific motion prediction. In *ECCV*, 2016.

[13] Somrita Banerjee, James Harrison, P Michael Furlong, and Marco Pavone. Adaptive meta-learning for identification of rover-terrain dynamics. *ArXiv*, 2020.

[14] Moshe Bar. The proactive brain: memory for predictions. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 2009.

[15] 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.

[16] 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.

[17] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. In *ECCV*, 2016.

[18] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *CoRL*, 2022.

[19] Josh C. Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 2006.

[20] Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *ICRA*, 2014.

[21] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

[22] Andreja Bubic, D Yves Von Cramon, and Ricarda I Schubotz. Prediction, cognition and the brain. *Frontiers in human neuroscience*, 4:25, 2010.

[23] Lars Buesing, Théophane Weber, Sébastien Racanière, S. M. Ali Eslami, Danilo Jimenez Rezende, David P. Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. Learning and querying fast generative models for reinforcement learning. *ArXiv*, 2018.

[24] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019.

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

[26] Lluís Castrejón, Nicolas Ballas, and Aaron C. Courville. Improved vrnns for video prediction. In *ICCV*, 2019.

[27] Fu-Hsiang Chan, Yu-Ting Chen, Yu Xiang, and Min Sun. Anticipating accidents in dashcam videos. In *ACCV*, 2017.

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

[29] Yu-Wei Chao, Jimei Yang, Brian L. Price, Scott Cohen, and Jia Deng. Forecasting human dynamics from static images. In *CVPR*, 2017.

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

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

[32] Prithvijit Chattopadhyay, Judy Hoffman, Roozbeh Mottaghi, and Aniruddha Kembhavi. Robustnav: Towards benchmarking robustness in embodied navigation. In *ICCV*, 2021.

[33] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *ICML*, 2017.

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

[35] Rohan Chitnis, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Learning quickly to plan quickly using modular meta-learning. In *ICRA*, 2019.

[36] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS Workshop on Deep Learning*, 2014.

[37] Andy Clark. Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and brain sciences*, 36(3):181–204, 2013.

[38] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 2015.

[39] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Eco: Efficient convolution operators for tracking. In *CVPR*, 2017.

[40] Samyak Datta, Oleksandr Maksymets, Judy Hoffman, Stefan Lee, Dhruv Batra, and Devi Parikh. Integrating egocentric localization for more realistic point-goal navigation agents. In *CoRL*, 2021.

[41] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[42] Matt Deitke, Dhruv Batra, Yonatan Bisk, Tommaso Campari, Angel X Chang, Devendra Singh Chaplot, Changan Chen, Claudia Pérez D'Arpino, Kiana Ehsani, Ali Farhadi, et al. Retrospectives on the embodied ai workshop. In *ArXiv*, 2022.

[43] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. Robothor: An open simulation-to-real embodied ai platform. In *CVPR*, 2020.

[44] Matt Deitke, Rose Hendrix, Luca Weihs, Ali Farhadi, Kiana Ehsani, and Aniruddha Kembhavi. Phone2proc: Bringing robust robots into our chaotic world. In *CVPR*, 2023.

[45] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. In *NeurIPS*, 2022.

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[47] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *ICLR*, 2017.

[48] Danny Driess, Ingmar Schubert, Pete Florence, Yunzhu Li, and Marc Toussaint. Reinforcement learning with neural radiance fields. In *NeurIPS*, 2022.

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

[50] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

[51] 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.

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

[53] Kiana Ehsani, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Object manipulation via visual target localization. In *ECCV*, 2022.

[54] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. In *CVPR*, 2021.

[55] Kiana Ehsani, Shubham Tulsiani, Saurabh Gupta, Ali Farhadi, and Abhinav Gupta. Use the force, luke! learning to predict physical forces by simulating effects. In *CVPR*, 2020.

[56] Ben Evans, Abitha Thankaraj, and Lerrel Pinto. Context is everything: Implicit identification for dynamics adaptation. In *ICRA*, 2022.

[57] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. In *Science Robotics*, 2020.

[58] Kuan Fang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Learning generalizable skills via automated generation of diverse tasks. In *RSS*, 2021.

[59] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *ICCV*, 2017.

[60] V Feinberg, A Wan, I Stoica, MI Jordan, JE Gonzalez, and S Levine. Model-based value expansion for efficient model-free reinforcement learning. In *ICML*, 2018.

[61] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[62] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *NeurIPS*, 2016.

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

[64] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, 2016.

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

[66] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *ICCV*, 2015.

[67] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. In *NeurIPS Dataset*, 2021.

[68] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel LK Yamins, James J DiCarlo, Josh McDermott, Antonio Torralba, et al. The threedworld transport challenge: A visually guided task-and-motion planning benchmark towards physically realistic embodied ai. In *ICRA*, 2022.

[69] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017.

[70] Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. Navigating to objects in the real world. In *ArXiv*, 2022.

[71] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

[72] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

[73] Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A Pires, and Rémi Munos. Neural predictive belief representations. In *ArXiv*, 2018.

[74] Zhaohan Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *ICML*, 2020.

[75] Abhishek Gupta, Russell Mendonca, Yuxuan Liu, P. Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *NeurIPS*, 2018.

[76] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

[77] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

[78] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[79] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

[80] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *ArXiv*, 2018.

[81] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *ICLR*, 2021.

[82] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. In *ArXiv*, 2023.

[83] David Hall, Ben Talbot, Suman Raj Bista, Haoyang Zhang, Rohan Smith, Feras Dayoub, and Niko Sünderhauf. The robotic vision scene understanding challenge. In *ArXiv*, 2020.

[84] Nathan Hatch and Byron Boots. The value of planning for infinite-horizon model predictive control. In *ICRA*, 2021.

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

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

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

[88] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NeurIPS*, 2015.

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

[90] Noriaki Hirose, Amir Sadeghian, Marynel Vázquez, Patrick Goebel, and Silvio Savarese. Gonet: A semi-supervised deep learning approach for traversability estimation. In *IROS*, 2018.

[91] Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep visual mpc-policy learning for navigation. In *Advanced Robotics*, 2019.

[92] Zhang-Wei Hong, Chen Yu-Ming, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, Hsuan-Kung Yang, Brian Hsi-Lin Ho, Chih-Chieh Tu, Yueh-Chuan Chang, Tsu-Ching Hsiao, et al. Virtual-to-real: Learning to control in visual semantic segmentation. In *IJCAI*, 2018.

[93] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *CVPR*, 2019.

[94] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *CoRL*, 2022.

[95] Yaohui Huang, Matthias Hofer, and Raffaello D'Andrea. Offset-free model predictive control: A ball catching application with a spherical soft robotic arm. In *IROS*, 2021.

[96] Carlo Innamorati, Bryan Russell, Danny M Kaufman, and Niloy J Mitra. Neural re-simulation for generating bounces in single images. In *ICCV*, 2019.

[97] Unnat Jain, Luca Weihs, Eric Kolve, Mohammad Rastegari, Svetlana Lazebnik, Ali Farhadi, Alexander G Schwing, and Aniruddha Kembhavi. Two body problem: Collaborative visual task completion. In *CVPR*, 2019.

[98] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *NeurIPS*, 2019.

[99] 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.

[100] 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.

[101] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *NeurIPS*, 2021.

[102] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings for embodied ai. In *CVPR*, pages 14829–14838, 2022.

[103] Seungsu Kim, Ashwini Shukla, and Aude Billard. Catching objects in flight. *IEEE Transactions on Robotics*, 30:1049–1065, 2014.

[104] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv*, 2014.

[105] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. Activity forecasting. In *ECCV*, 2012.

[106] 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. *ArXiv*, 2017.

[107] 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.

[108] Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *EMNLP*, 2020.

[109] Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *EMNLP*, 2020.

[110] 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.

[111] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. In *RSS*, 2021.

[112] Ashish Kumar, Zhongyu Li, Jun Zeng, Deepak Pathak, Koushil Sreenath, and Jitendra Malik. Adapting rapid motor adaptation for bipedal robots. In *IROS*, 2022.

[113] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *ICLR*, 2018.

[114] Tian Lan, Tsung-Chuan Chen, and Silvio Savarese. A hierarchical representation for future action prediction. In *ECCV*, 2014.

[115] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In *ICML*, 2020.

[116] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B. Choy, Philip H. S. Torr, and Manmohan Chandraker. DESIRE: distant future prediction in dynamic scenes with interacting agents. In *CVPR*, 2017.

[117] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *ArXiv*, 2016.

[118] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NeurIPS*, 2014.

[119] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, 2013.

[120] Thomas Lew, Apoorva Sharma, James Harrison, Andrew Bylard, and Marco Pavone. Safe active dynamics learning and control: A sequential exploration–exploitation framework. *IEEE Transactions on Robotics*, 2022.

[121] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. In *CoRL*, 2021.

[122] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *CoRL*, 2022.

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

[124] Shangda Li, Devendra Singh Chaplot, Yao-Hung Hubert Tsai, Yue Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov. Unsupervised domain adaptation for visual navigation. *ArXiv*, 2020.

[125] Shunkai Li, Xin Wang, Yingdian Cao, Fei Xue, Zike Yan, and Hongbin Zha. Self-supervised deep visual odometry with online adaptation. In *CVPR*, 2020.

[126] 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.

[127] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[128] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[129] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. In *Science Robotics*, 2021.

[130] Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. Unified-io: A unified model for vision, language, and multi-modal tasks. In *ICLR*, 2023.

[131] Lucas Manuelli, Yunzhu Li, Pete Florence, and Russ Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning. In *CoRL*, 2020.

[132] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016.

[133] R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *ACL*, 2019.

[134] Antoine Miech, Ivan Laptev, Josef Sivic, Heng Wang, Lorenzo Torresani, and Du Tran. Leveraging the present to anticipate the future in videos. In *CVPR Workshops*, 2019.

[135] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. In *NeurIPS*, 2018.

[136] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2017.

[137] 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.

[138] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[139] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NeurIPS Workshop*, 2013.

[140] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[141] Jun Morimoto and Kenji Doya. Robust reinforcement learning. In *NeurIPS*, 2001.

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

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

[144] Mark Muller, Sergei Lupashin, and Raffaello D'Andrea. Quadrocopter ball juggling. In *IROS*, 2011.

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

[146] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, P. Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *ICLR*, 2018.

[147] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018.

[148] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018.

[149] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. In *CVPR*, 2016.

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

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

[152] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[153] Tianwei Ni, Kiana Ehsani, Luca Weihs, and Jordi Salvador. Towards disturbance-free visual mobile manipulation. In *WACV*, 2023.

[154] OpenAI. Chatgpt: Optimizing language models for dialogue. In *OpenAI blog*, 2019.

[155] Michael O'Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural-fly enables rapid learning for agile flight in strong winds. In *Science Robotics*, 2022.

[156] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *AAAI*, 2022.

[157] Hyun Soo Park, Jyh-Jing Hwang, Yedong Niu, and Jianbo Shi. Egocentric future localization. In *CVPR*, 2016.

[158] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

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

[160] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *ICCV*, 2019.

[161] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *RSS*, 2020.

[162] Silvia L. Pintea, Jan C. van Gemert, and Arnold W. M. Smeulders. Déjà vu: Motion prediction in static images. In *ECCV*, 2014.

[163] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *ICML*, 2017.

[164] Senthil Purushwalkam, Abhinav Gupta, Danny M Kaufman, and Bryan Russell. Bounce and learn: Modeling scene dynamics with real-world bounces. In *ICLR*, 2019.

[165] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *NeurIPS*, 2017.

[166] Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *NeurIPS*, 2017.

[167] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[168] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. In *OpenAI*, 2018.

[169] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. In *OpenAI blog*, 2019.

[170] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. In *ICLR*, 2017.

[171] Nicholas Rhinehart and Kris M. Kitani. First-person activity forecasting with online inverse reinforcement learning. In *ICCV*, 2017.

[172] Robin Ritz, Mark W. Müller, Markus Hehn, and Raffaello D'Andrea. Cooperative quadrocopter ball throwing and catching. In *IROS*, 2012.

[173] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[174] Fereshteh Sadeghi and Sergey Levine. CAD2RL: real single-image flight without a single real image. In *RSS*, 2017.

[175] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[176] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018.

[177] 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.

[178] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

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

[180] Neal Seegmiller, Forrest Rogers-Marcovitz, Greg Miller, and Alonzo Kelly. Vehicle model identification by integrated prediction error minimization. *The Intl. J. of Robotics Research*, 2013.

[181] Carmelo Sferrazza, Michael Muehlebach, and Raffaello D'Andrea. Learning-based parametrized model predictive control for trajectory tracking. In *Optimal Control Applications and Methods*, 2020.

[182] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *CoRL*, 2022.

[183] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *CVPR*, 2020.

[184] Rui Silva, Francisco S. Melo, and Manuela M. Veloso. Towards table tennis with a quadrotor autonomous learning robot and onboard vision. In *IROS*, 2015.

[185] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 2017.

[186] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *ICML*, 2017.

[187] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *IROS*, 2020.

[188] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015.

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

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

[191] Kunyue Su and Shaojie Shen. Catching a flying ball with a vision-based quadrotor. In *ISER*, 2017.

[192] Chen Sun, Abhinav Shrivastava, Carl Vondrick, Rahul Sukthankar, Kevin Murphy, and Cordelia Schmid. Relational action forecasting. In *CVPR*, 2019.

[193] Chong Sun, Huchuan Lu, and Ming-Hsuan Yang. Learning spatial-aware regressions for visual tracking. In *CVPR*, 2018.

[194] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 2000.

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

[196] Tomoyuki Suzuki, Hirokatsu Kataoka, Yoshimitsu Aoki, and Yutaka Satoh. Anticipating traffic accidents with adaptive loss and large-scale incident db. In *CVPR*, 2018.

[197] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. In *NeurIPS*, 2021.

[198] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *NeurIPS*, 2016.

[199] J. K Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sulivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. In *ArXiv*, 2020.

[200] 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.

[201] Manuela M. Veloso, Elly Winner, Scott Lenser, James Bruce, and Tucker R. Balch. Vision-servoed localization and behavior-based planning for an autonomous quadruped legged robot. In *AIPS*, 2000.

[202] Vandi Verma, Geoffrey J. Gordon, Reid G. Simmons, and Sebastian Thrun. Real-time fault diagnosis [robot fault diagnosis]. *IEEE Robotics & Automation Magazine*, 2004.

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

[204] Ruben Villegas, Arkanath Pathak, Harini Kannan, Dumitru Erhan, Quoc V. Le, and Honglak Lee. High fidelity video prediction with large stochastic recurrent neural networks. In *NeurIPS*, 2019.

[205] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *ICLR*, 2017.

[206] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating visual representations from unlabeled video. In *CVPR*, 2016.

[207] Nolan Wagener, Ching-An Cheng, Jacob Sacks, and Byron Boots. An online learning approach to model predictive control. In *RSS*, 2019.

[208] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *ECCV*, 2016.

[209] Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014.

[210] Jacob Walker, Abhinav Gupta, and Martial Hebert. Dense optical flow prediction from a static image. In *ICCV*, 2015.

[211] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. The pose knows: Video forecasting by generating pose futures. In *ICCV*, 2017.

[212] Matthew Wallingford, Aditya Kusupati, Alex Fang, Vivek Ramanujan, Aniruddha Kembhavi, Roozbeh Mottaghi, and Ali Farhadi. Neural radiance field codebooks. In *ICLR*, 2023.

[213] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *ICLR*, 2018.

[214] Tsun-Hsuan Wang, Hung-Jui Huang, Juan-Ting Lin, Chan-Wei Hu, Kuo-Hao Zeng, and Min Sun. Omnidirectional cnn for visual place recognition and navigation. In *ICRA*, 2018.

[215] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *ECCV*, 2018.

[216] Saim Wani, Shivansh Patel, Unnat Jain, Angel X. Chang, and Manolis Savva. Multion: Benchmarking semantic map memory using multi-object navigation. In *NeurIPS*, 2020.

[217] 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.

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

[219] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *CVPR*, 2021.

[220] 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.

[221] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[222] 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.

[223] 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.

[224] 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.

[225] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *NeurIPS*, 2017.

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

[227] 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.

[228] 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.

[229] Dan Xie, Sinisa Todorovic, and Song-Chun Zhu. Inferring "dark matter" and "dark energy" from videos. In *ICCV*, 2013.

[230] 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.

[231] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NeurIPS*, 2016.

[232] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. In *ICLR*, 2019.

[233] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. In *ICLR*, 2019.

[234] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Dee M, Jodilyn Peralta, Brian Ichter, Karol Hausman, and Fei Xia. Scaling robot learning with semantically imagined experience. In *ArXiv*, 2023.

[235] Wenhao Yu, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *RSS*, 2017.

[236] Wenhao Yu, C. Karen Liu, and Greg Turk. Policy transfer with strategy optimization. In *ICLR*, 2019.

[237] Wenhao Yu, Jie Tan, Yunfei Bai, Erwin Coumans, and Sehoon Ha. Learning fast adaptation with meta strategy optimization. *IEEE Robotics and Automation Letters*, 2020.

[238] Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *RSS*, 2018.

[239] Jenny Yuen and Antonio Torralba. A data-driven approach for event prediction. In *ECCV*, 2010.

[240] Kuo-Hao Zeng, Shih-Han Chou, Fu-Hsiang Chan, Juan Carlos Niebles, and Min Sun. Agent-centric risk assessment: Accident anticipation and risky region localization. In *CVPR*, 2017.

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

[242] 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.

[243] Kuo-Hao Zeng, Luca Weihs, Ali Farhadi, and Roozbeh Mottaghi. Pushing it out of the way: Interactive visual navigation. In *CVPR*, 2021.

[244] Dingqi Zhang, Antonio Loquercio, Xiangyu Wu, Ashish Kumar, Jitendra Malik, and Mark W Mueller. A zero-shot adaptive quadcopter controller. In *ArXiv*, 2022.

[245] Yuan Zhang, Jason Baldridge, and Luheng He. Paws: Paraphrase adversaries from word scrambling. In *NAACL*, 2019.

[246] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. In *ArXiv*, 2023.

[247] Bo Zheng, Yibiao Zhao, Joey C. Yu, Katsushi Ikeuchi, and Song-Chun Zhu. Scene understanding by reasoning stability and safety. *IJCV*, 2014.

[248] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. In *ICLR*, 2019.

[249] Wenxuan Zhou, Lerrel Pinto, and Abhinav Kumar Gupta. Environment probing interaction policies. In *ICLR*, 2019.

[250] 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.

[251] 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.

[252] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.